



EVALUATION OF A FIELD PROGRAMMABLE GATE ARRAY
CIRCUIT RECONFIGURATION SYSTEM

THESIS

Jason L. Ives, Captain, USAF

AFIT/GE/ENG/06-26

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

EVALUATION OF A FIELD PROGRAMMABLE GATE ARRAY
CIRCUIT RECONFIGURATION SYSTEM

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Electrical Engineering

Jason L. Ives, B.S.E.E.
Captain, USAF

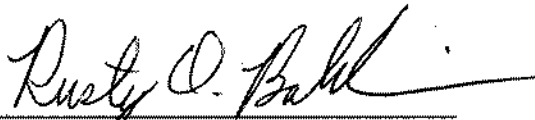
March 2006

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

EVALUATION OF A FIELD PROGRAMMABLE GATE ARRAY
CIRCUIT RECONFIGURATION SYSTEM

Jason L. Ives, B.S.E.E.
Captain, USAF

Approved:



Dr. Rusty O. Baldwin (Chairman)

2 Mar 06

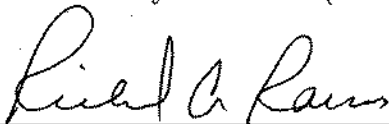
date



Dr. Barry E. Mullins (Member)

3 Mar 06

date



Dr. Richard A. Raines (Member)

3 Mar 06

date

Abstract

This research implements a circuit reconfiguration system (CRS) to reconfigure a field programmable gate array (FPGA) in response to a faulty configurable logic block (CLB). It is assumed the location of the fault is known and the CLB is moved according to one of four replacement methods: column left, column right, row up, and row down. Partial reconfiguration of the FPGA is done through the JTAG port to produce the desired logic block movement. The time required to accomplish the reconfiguration is measured for each method in both clear and congested areas of the FPGA. The measured data indicates there is no consistently better replacement method regardless of the circuit congestion or location within the FPGA. Thus, given a specific location in the FPGA, there is no preferred replacement method that will result in the lowest reconfiguration time.

Acknowledgements

First, I would like to thank my wife for keeping me clothed and fed during my many long days at AFIT and for supporting me 100% in everything I do. Second, I want to thank my sons for understanding when I couldn't devote as much time to them as they wanted. They've made the greatest sacrifice of anyone involved in this endeavor - they sacrificed time with their dad. I would also like to thank my colleague and friend, LeRoy Willemsen, for his valuable insight and feedback on my work and for helping me through the rough spots. Finally, I want to thank my thesis adviser, Dr. Rusty Baldwin, for his unending enthusiasm, optimism and patience. I would have been lost in the woods without his guidance.

Jason L. Ives

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
List of Figures	viii
List of Tables	x
List of Abbreviations	xi
I. Introduction	1
1.1 Overview	1
1.2 Motivation and Goals	1
1.3 Organization	2
II. Literature Review	3
2.1 Fault Tolerant Computing	3
2.2 General Fault Tolerance	4
2.2.1 Fault Tolerant Strategies	4
2.2.2 Redundancy	6
2.3 Field Programmable Gate Arrays	6
2.3.1 Advantages of FPGAs	10
2.3.2 Disadvantages of FPGAs	10
2.4 Fault Tolerance with Field Programmable Gate Arrays	11
2.4.1 Yield Enhancement	12
2.4.2 Reconfiguration	15
2.4.3 Performance of Reconfiguration Schemes	17
2.5 Summary	18
III. Research Methodology	19
3.1 Problem Definition	19
3.1.1 Goals and Hypothesis	19
3.1.2 Approach	19
3.2 System Boundaries	20
3.3 System Services	21
3.4 Workload	21
3.5 Performance Metrics	22
3.6 Parameters	22

	Page
3.6.1 System	22
3.6.2 Workload	23
3.7 Factors	23
3.8 Evaluation Technique	24
3.9 Experimental Design	24
3.10 Results Analysis	25
3.11 Summary	26
IV. Data Analysis	27
4.1 Introduction	27
4.2 Validation	27
4.3 Initial Analysis	28
4.3.1 Analysis by Location Type and Method	28
4.3.2 Comparison of Observations with Hypotheses	33
4.4 Second Analysis	35
4.4.1 Analysis of Clear Locations	35
4.4.2 Analysis of Congested Locations	36
4.4.3 Analysis by Specific Location	40
4.4.4 Comparison of Results of Second Analysis and Hypotheses	42
4.5 Summary	45
V. Conclusions	47
5.1 Introduction	47
5.2 Problem Summary	47
5.3 Conclusions of Research	47
5.4 Significance of Research	48
5.5 Recommendations for Future Research	49
5.5.1 Other FPGAs and Programming Interfaces	49
5.5.2 Automation	49
5.5.3 Bit File Manipulation	49
Appendix A. Experimental Configuration	50
Appendix B. Data	56
Bibliography	59
Vita	63
Index	63
Author Index	63

List of Figures

Figure		Page
2.1	Redundancy Techniques	7
2.2	Basic structure of an FPGA	8
2.3	Basic structure of a CLB	9
3.1	System Boundaries of the Circuit Reconfiguration System . . .	21
4.1	Plot of Measured Reconfiguration Times	27
4.2	Residuals versus Fitted Values	29
4.3	Normal Probability Plot of Residuals	29
4.4	Confidence Interval Plot of Reconfiguration Time versus Method	30
4.5	Confidence Interval Plot of Reconfiguration Time versus Method Type	30
4.6	Confidence Interval Plot of Reconfiguration Time versus Method Type for Combined Location Types	31
4.7	Confidence Interval Plot of Reconfiguration Time versus Method, Location Type	32
4.8	Confidence Interval Plot of Reconfiguration Time versus Loca- tion Type	33
4.9	Confidence Interval Plot of Reconfiguration Time versus Loca- tion Type	34
4.10	Residuals versus Fitted Values for Clear Location Type	35
4.11	Normal Probability Plot of Residuals for Clear Location Type .	36
4.12	Confidence Interval Plot of Reconfiguration Time for Clear Lo- cations and Column Left Method	37
4.13	Confidence Interval Plot of Reconfiguration Time for Clear Lo- cations and Column Right Method	37
4.14	Confidence Interval Plot of Reconfiguration Time for Clear Lo- cations and Row Up Method	38

Figure		Page
4.15	Confidence Interval Plot of Reconfiguration Time for Clear Locations and Row Down Method	38
4.16	Residuals versus Fitted Values for Congested Location Type . .	39
4.17	Normal Probability Plot of Residuals for Congested Location Type	39
4.18	Confidence Interval Plot of Reconfiguration Time for Congested Locations and Column Left Method	40
4.19	Confidence Interval Plot of Reconfiguration Time for Congested Locations and Column Right Method	41
4.20	Confidence Interval Plot of Reconfiguration Time for Congested Locations and Row Up Method	41
4.21	Confidence Interval Plot of Reconfiguration Time for Congested Locations and Row Down Method	42
4.22	Confidence Interval Plot of Reconfiguration Time versus Replacement Method for Clear Locations	43
4.23	Confidence Interval Plot of Reconfiguration Time versus Replacement Method for Congested Locations	43
4.24	Confidence Interval Plot of Reconfiguration Time versus Replacement Method Type for Clear Locations	44
4.25	Confidence Interval Plot of Reconfiguration Time versus Replacement Method Type for Congested Locations	44
A.1	Schematic of the LUT and State Logic in the Circuit Reconfiguration System	50
A.2	Post Configuration Processor of the Circuit Reconfiguration System	51
A.3	Schematic of the Timer Circuit used to Measure the Circuit Reconfiguration System	53

List of Tables

Table		Page
3.1	Factors and Levels for the Circuit Reconfiguration System . . .	23
3.2	Cross Reference of Location Names to FPGA Slice Numbers . .	25
B.1	Measured Reconfiguration Times for Location A1	56
B.2	Measured Reconfiguration Times for Location A2	56
B.3	Measured Reconfiguration Times for Location A3	57
B.4	Measured Reconfiguration Times for Location B1	57
B.5	Measured Reconfiguration Times for Location B2	57
B.6	Measured Reconfiguration Times for Location B3	58

List of Abbreviations

Abbreviation		Page
FPGA	Field Programmable Gate Array	1
CRS	Circuit Reconfiguration System	1
IC	Integrated Circuit	3
MTBF	Mean Time Between Failures	4
ALU	Arithmetic Logic Unit	5
ASIC	Application Specific Integrated Circuits	6
CLB	Configurable Logic Block	7
LUT	Lookup Table	8
SUT	System Under Test	20
CUT	Component Under Test	20
JTAG	Joint Test Action Group	24
TCK	Test Clock	24
ICAP	Internal Configuration Access Port	49

EVALUATION OF A FIELD PROGRAMMABLE GATE ARRAY CIRCUIT RECONFIGURATION SYSTEM

I. Introduction

1.1 *Overview*

In modern computer and electronic systems, the occurrence of faults is nearly unavoidable at some point in the system's lifetime. These faults can be caused by many factors including defects at the device level.

Fault tolerant methods detect and reconfigure in the presence of system faults so the system can function even if it's at a reduced efficiency or capability. The ability of a field programmable gate array (FPGA) to reconfigure makes it an effective device for implementing fault tolerance through circuit reconfiguration. Fault tolerance applied to an FPGA detects faults while the user's circuit is operating and then replaces faulty components of the circuit by reconfiguring the FPGA.

1.2 *Motivation and Goals*

The ability to operate in the presence of defects and faults will greatly benefit digital systems. Currently, spacecraft largely rely on redundancy to ensure continued operation. A fault and defect tolerant system able to reconfigure itself in response to faults would, at the very least, reduce the dependence on redundant systems [LCR03], perhaps freeing up area on the spacecraft for other systems. This research will also benefit other Air Force aircraft and weapons systems where high reliability is required or system maintenance is difficult or impossible.

The goal of this research is to develop a circuit reconfiguration system (CRS) with four replacement methods to reconfigure an FPGA.

1.3 Organization

Chapter 1 provides an introduction to this research and explains the motivation for the study. Chapter 2 presents background information on fault tolerance and FPGAs as well as a review of the literature pertinent to fault tolerance using FPGAs. Chapter 3 describes the research methodology employed in this study while Chapter 4 analyzes the measured data. Chapter 5 contains the conclusions of this study and suggests areas for future research followed by an appendix describing the experimental setup used.

II. Literature Review

2.1 *Fault Tolerant Computing*

In military and aerospace applications, fault tolerant systems increase reliability and reduce maintenance. Even the most carefully designed and implemented system has some probability of failure, however small it may be. The ability to detect or mask faults and errors is critical to fault tolerance and can further increase system reliability.

The goal of fault tolerance is to improve system dependability by enabling a system to operate correctly in the presence of faults [Nel90]. Dependability, or reliability, is characterized by the probability of a system being able to perform as specified over a certain time period [AL81]. Dependable systems are of great importance in environments requiring high reliability or where it is impractical or difficult to perform maintenance, such as in space.

Fault tolerant computing techniques emerged in the early years of computer development [Nei03]. Computers were built from large, bulky, and largely unreliable components connected together with thick wires and solder joints. Among the thousands of components and connections in the machine, it was inevitable that at least one would prove faulty. Exacerbating the problem, some faults were intermittent making them harder to locate. The system impact of the various faults also varied. A transient fault could cause a serious performance degradation or halt the system altogether while a permanent fault would curiously have little effect on the system. Engineers realized it was futile to try to build a fault-free system and instead devised ways for the system to operate in spite of the faults.

As technology advanced, and certainly with the introduction of solid state devices and integrated circuits (ICs), device reliability improved and interest in fault tolerance declined. However, interest in fault tolerance is returning. Device size on ICs has shrunk dramatically with a corresponding increase in faults. This is leading to a renewed interest in fault tolerance techniques [NSF01].

2.2 General Fault Tolerance

In the domain of fault tolerance, failures, faults, and errors are distinct terms even though in many contexts, they are used interchangeably. A fault is the physical malfunction of a system caused by a defective component, physical damage, or design error. A failure is the inability of a component, circuit, or element to perform its function due to errors [Nel90]. An error is the realization of a fault evidenced by incorrect data or output from the component. A fault does not necessarily cause an error and failure may or may not occur as the result of an error. Improving the ability of a system to tolerate a certain number of faults can be accomplished by preventing, or at least masking, errors caused by the faults. The ability to prevent or mask errors reduces failures and increases reliability.

A fault tolerant system isn't necessarily highly reliable nor does a reliable system necessarily incorporate fault tolerance [Nel90]. Fault tolerance is simply one aspect of reliability. Many other factors contribute to overall system reliability. Defining reliability can be an elusive task. If a system provides incorrect data but is performing according to its design and specifications, it can't be deemed unreliable. In fact, it will *reliably* provide incorrect data unless its specifications or design are changed. The true reliability of a system can't be predicted precisely, but there are a number of widely accepted measures of reliability, a common one being the Mean Time Between Failures (MTBF), which is the average time a system operates before failing.

2.2.1 Fault Tolerant Strategies. To design a fault tolerant system, a fault tolerance strategy must be formulated that includes one or more of the following elements [Kwi97]:

- **Masking:** Correction of errors.
- **Detection:** Detection of errors.
- **Containment:** Prevention of error propagation through the system.

- **Diagnosis:** Location and identification of a faulty module or component responsible for an error.
- **Repair/reconfiguration:** Elimination or replacement of a faulty module.
- **Recovery:** Return of the system to acceptable operation.

The particular application is a driving force in the development of a fault tolerance strategy and which elements are included. Some of the elements, such as masking and detection, are complimentary and it would clearly be beneficial for both elements to be used together.

Fault and error modeling occurs at several levels of abstraction within the system [Nel90]. At higher levels, modeling and analysis become easier, but there is a corresponding decrease in accuracy. At the lowest level, faults occur in devices and are driven by the device manufacturing technology. At this level, physical defects such as shorts or opens cause erroneous voltages, incorrect currents or mistimed switching. This level of fault and error modeling can accurately determine the cause of errors, but it also can make it difficult to pinpoint the defective device at a microscopic level.

The next level of abstraction is the logical level. At this level, a system is modeled as various gates and memory elements [Nel90]. This level represents device outputs as binary values at specific points in the circuit. Faults modeled at the logical level include “stuck-at” faults and incorrect or missing inputs to gates. There is a slight loss of accuracy at this level in that a fault is not pinpointed down to a specific component but it allows the designer to evaluate the binary signals within the circuit and more easily analyze circuit performance.

A third level of abstraction combines gates into a unit with a function such as a register, arithmetic logic unit (ALU), or processor. Accuracy is traded for ease of simulating module behavior. At this level, a module’s behavior is evaluated to determine if a fault exists by examining state or truth tables.

2.2.2 Redundancy. Redundancy corrects errors caused by faults and prevents propagation of errors to other parts of the system [Kwi97]. Redundancy is the key ingredient in achieving fault tolerance [SP03, CC95]. Shown in Figure 2.1 are the four basic redundancy techniques for fault tolerance: redundant information, computations, software and hardware [AL81].

- Information redundancy includes additional information, such as error detection and error correction codes, for handling faults [McF94]. Some commonly used codes are parity codes, checksums, cyclic redundancy codes, and Hamming codes.
- Computational (or time) redundancy repeats computations to detect and tolerate faults [Kwi97]. This is an especially effective technique in the presence of transient faults. Two common methods are re-computation with shifted operands and re-computing with swapped operands.
- Software redundancy uses additional lines of code or small programs to detect and correct faults [McF94]. Result consistency and scaling is commonly checked and system capability is evaluated.
- Hardware redundancy includes spare hardware resources used in place of faulty modules [McF94]. There are three different types of hardware redundancy: passive, active, and hybrid. Passive techniques, such as N-modular redundancy and majority voting, mask faults while active techniques detect and locate faults and initiate system reconfiguration. Common active techniques include duplication with comparison and standby sparing. Hybrid redundancy combines passive and active techniques to detect and remove errors.

2.3 Field Programmable Gate Arrays

FPGAs were introduced in the early 1980s to bridge the gap between traditional programmable logic devices and application specific integrated circuits (ASIC) [Max04]. An FPGA has a unique combination of traits which allow it to implement

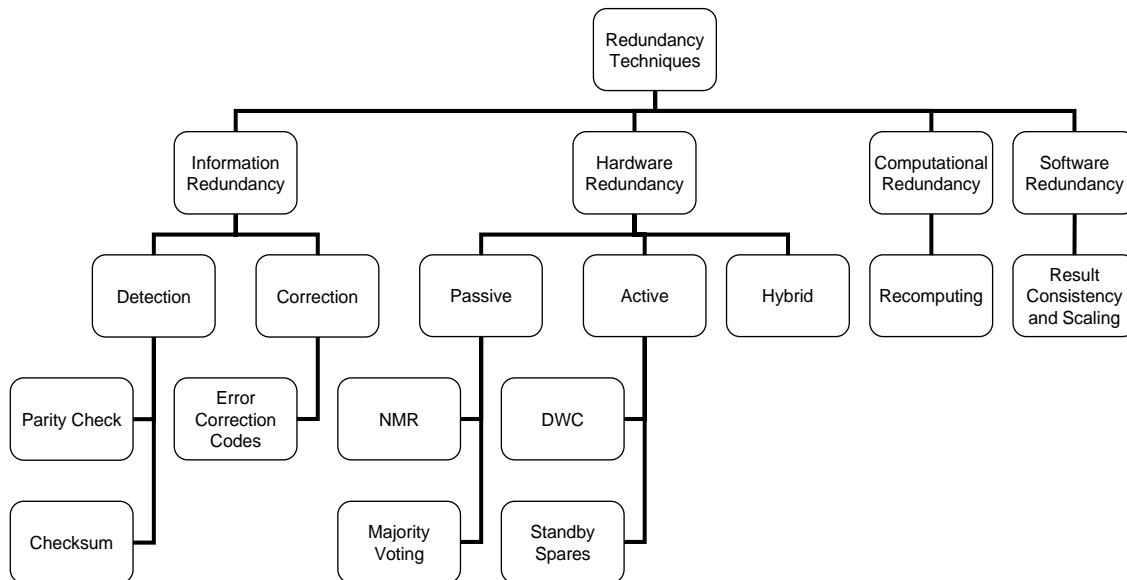


Figure 2.1: The four basic redundancy techniques are Information Redundancy, Hardware Redundancy, Computational Redundancy, and Software Redundancy [Pra05].

complex logic functions on a chip without the high cost or excessive design and fabrication time of an ASIC [Hau98]. FPGAs are particularly useful for rapid prototyping. A design can be implemented and evaluated quickly without the permanency of fabricating the circuit in silicon. Furthermore, the design can be changed and the FPGA can be reconfigured easily. Reconfigurability has led to the possibility of using FPGAs for fault tolerance.

FPGAs generally have a structure similar to Figure 2.2: an array of configurable logic blocks (CLBs) surrounded by horizontal and vertical routing channels [Tor02]. The edges of the array are populated with input/output blocks which provide access to the I/O pins of the chip. The routing channels are wire segments of varying lengths that connect logic blocks to I/O blocks as well as to other logic blocks. Intersection points of the routing channels have switch matrices which are programmed to properly route the wire segments. The I/O blocks are also programmable and can be configured to be compatible with various I/O interface standards [Max04]. An FPGA has a configuration memory to store a configuration bit file which determines how the CLBs in the FPGA are used to implement a particular circuit. Each unique circuit has a

unique bit file. Rearranging the bits in the file changes how the CLBs are configured resulting in a different circuit or a new placement of the same circuit.

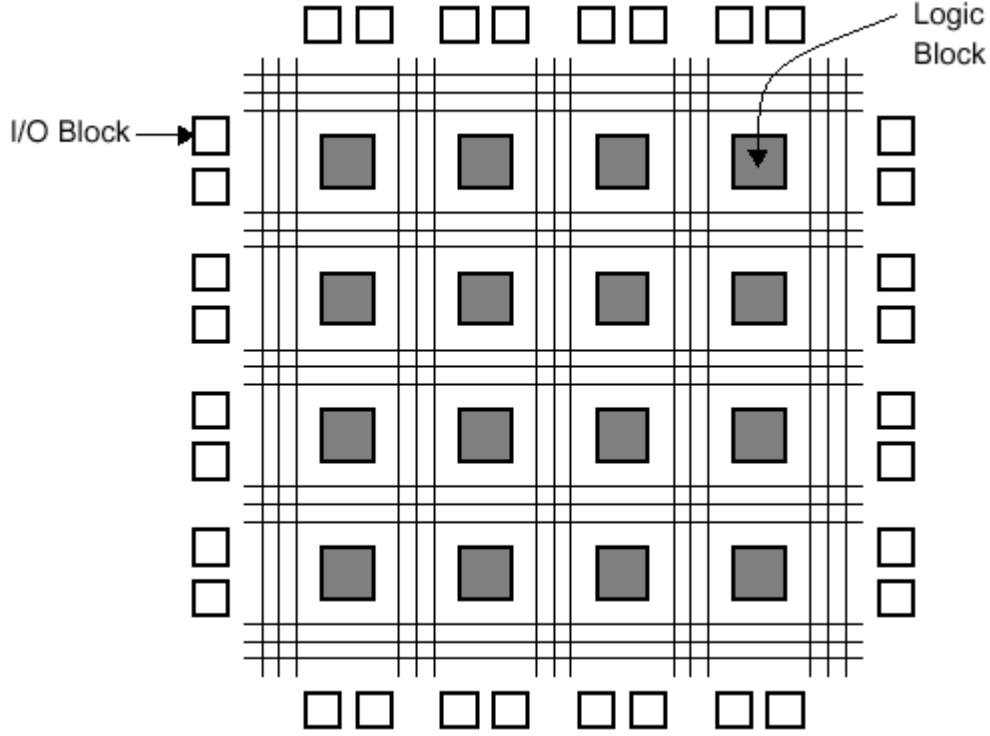


Figure 2.2: The basic structure of an FPGA is an array of CLBs with channels of routing resources between them. Switch matrices lie at the intersection points of the routing lines to make the proper connections between CLBs and the I/O blocks on the edges of the FPGA [Kha02].

A typical CLB, such as the one shown in Figure 2.3, consists of a 3- or 4-input lookup table (LUT), a multiplexer, and a register [Max04]; however, the specific configuration is device dependent. The LUT can be programmed to emulate any n -input logic function where n is the number of LUT input lines. The multiplexer selects between the output of the LUT and an input from outside the CLB. The register latches the logic value of the CLB or an input from outside the CLB for later use as a clocked output, q . The output y is the result of the logic operation programmed into the LUT and can serve as an input to other combinational logic in the array. Every CLB in the FPGA can be programmed to perform a different

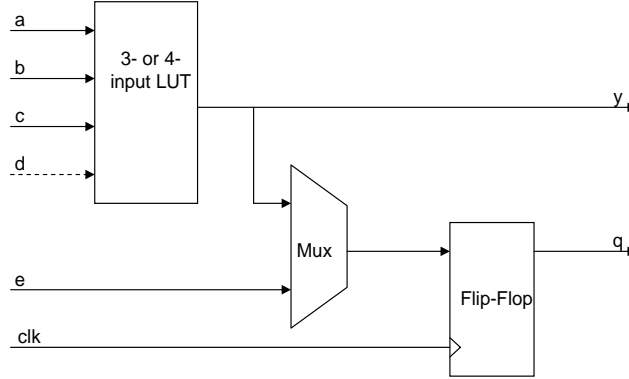


Figure 2.3: The basic structure of a CLB consists of a lookup table, a multiplexer, and a register [Max04].

logic function. The combination of CLBs forms some digital circuit that performs a desired function. This function could be a microprocessor, a digital signal processing module, or some other digital circuit as long as there are enough CLBs available in the FPGA [HKS98].

Granularity of the CLB is an important design consideration for FPGAs. Granularity is the amount of logic contained in the CLB and it determines the complexity of logic function that can be implemented in the CLB. A coarse-grained CLB contains more logic gates than a fine-grained CLB. Consequently, a coarse-grained CLB can implement more complex logic functions. The main advantage to using a fine-grained CLB is that, generally, the logic elements in the CLB are used more efficiently. Using a coarse-grained CLB to implement simple logic functions isn't as efficient as using a fine-grained CLB because some portion of the resources in the coarse-grained CLB won't be used [REGSV93]. The main drawback to using finer grained CLBs is the amount of interconnect resources required. As the granularity becomes finer, the number of connections into the blocks decreases, reducing delay and area costs. Coarse-grained CLBs generally reduce the number of blocks needed to implement a design.

Configuring an FPGA is done through configuration cells which determine connectivity between the device inputs and outputs and the CLBs and the connectivity

between CLBs. A configuration file is generated by a design tool based on schematics, a hardware design language, or other design flow and programs the FPGA to perform the desired function. The configuration file consists of configuration data and commands. Configuration data is the portion of the configuration file that defines the state of the programmable logic elements in the FPGA. Configuration commands specify how to use the configuration data [Max04]. Once created, a configuration file is loaded onto the FPGA by way of a configuration port.

2.3.1 Advantages of FPGAs. The main advantage of FPGAs is their capability to be reprogrammed. SRAM-based FPGAs can be reprogrammed a large but indeterminate number of times. Making design changes to a logic circuit implemented on an FPGA is simply a matter of generating a new configuration file and loading it onto the FPGA. The new circuit is then realized on the FPGA and ready for use. This versatility makes the FPGA ideal for applications where flexibility and design time is important.

Another advantage is that the complexity of the combinational logic function implemented on an FPGA is scalable [May97]. The same FPGA can be used to implement a range of circuits varying in the number of inputs, outputs, logic gates, and interconnections.

A third advantage is that some FPGAs can be partially reconfigured. This means part of the FPGA continues to operate while another area is programmed. The two circuits must be kept separate, but this capability allows the user to configure a new circuit, load it into the FPGA, and then switch operation over with minimal delay.

2.3.2 Disadvantages of FPGAs. One of the main problems with using FPGAs is signal propagation time. Unless carefully specified, interconnect length within FPGAs can vary for each configuration resulting in unpredictable signal delays. A circuit requires very careful placement and routing to ensure the signal propagation

characteristics are as expected and required. The architecture of an FPGA generally causes the placement and routing of a logic circuit to be different and possibly sub-optimal compared to how the circuit would be made in a traditional ASIC.

Another problem is chip area. FPGAs require more chip area than an ASIC to implement the same logic circuit due to the enormous amount of interconnect required to give the FPGA the flexibility to implement a wide array of different circuits. Over 90% of the chip area in most FPGAs is used for routing [CH02].

Finally, in terms of today's computing speeds, programming an FPGA is relatively slow. The larger the circuit and the more logic blocks and interconnects that must be programmed, the more time it takes. The method of programming also impacts programming time. Older FPGAs are programmed serially. The configuration bit file is loaded to the FPGA one bit at a time. Newer FPGAs can be programmed in parallel. Another way to decrease programming time is, as discussed above, through partial reconfigurability. If only a portion of the circuit needs to be changed, it makes sense to only reprogram that portion, which reduces programming time [May97]. In addition, if two different circuits will fit on a single FPGA, "hot" switch-overs are possible which also minimize the effect of programming delay.

2.4 Fault Tolerance with Field Programmable Gate Arrays

Fault tolerance using FPGAs should have the following goals [XSHL99]:

- Low overhead in terms of spare resources on the chip
- A simple, fast replacement algorithm
- The reconfigured circuit performs the same as the original circuit

The ability of an FPGA to reconfigure makes it an effective device for implementing fault tolerance through circuit reconfiguration. The goal of fault tolerance at this level is to detect faults while the circuit is operating and then replace the faulty components of the circuit [ESSA00]. Faults can occur in the logic of the circuit or in the routing between the CLBs. Spare resources must be set aside for use

when reconfiguration becomes necessary. To provide spare logic, a spare component must be defined at some level of granularity. Spare logic could mean a spare row of CLBs, column of CLBs, individual CLBs, or other logic block as determined by the replacement scheme.

In general, there are two similar approaches in current research on fault tolerance using FPGAs. One uses off-line techniques to improve the production yield of FPGA chips. The other approach takes advantage of the FPGA's regular cell structure to dynamically reconfigure a circuit on the chip [HTA94]. It's possible that yield enhancement techniques can be adapted for online reconfiguration and vice versa. In fact, there are many similarities between the two approaches.

Tolerating faults in the interconnect routing is more difficult than faults in the logic. Routing resources are limited and allocating spares isn't possible in some cases. A circuit implemented on an FPGA with a large number of CLBs using all of their input and output pins will have very few left over sections of interconnect to use as spares. In addition, the granularity of the CLBs, in part, determines the amount of routing available in the FPGA. A typical FPGA contains a much larger percentage of interconnect than CLBs making interconnect faults more likely.

2.4.1 Yield Enhancement. Improving the yield of FPGAs, and integrated circuits in general, has become increasingly important as device sizes have shrunk. The higher the density of devices on a chip, the higher the likelihood of a defect. Many chips become unusable if there is but a single defective device among the millions of devices on the chip.

Defects in an FPGA can occur in the devices making up the CLBs, the routing segments, or both. There are various techniques to deal with either type [MHS⁺04]. The challenge in overcoming a routing defect is to maintain signal propagation delay and integrity of the original circuit with a new routing scheme that uses different resources and may include longer interconnecting routes than previously used.

The first step in yield enhancement is to test an un-programmed device to locate defective components and populate a defect database [KDFJ89] which maps the device avoiding defective areas. The device is then programmed by placing the circuit according to the map. The process occurs at the time of manufacture and is transparent to the user.

There are a number of ways of obtaining a usable chip depending on how the FPGA architecture is constructed. One method arranges logic gates in the FPGA into quadrangles called “courtyards.” Courtyards are arranged in a rectangular array with the area between the courtyards called “streets” of interconnect [KDFJ89]. An “ideal” layout is constructed which assumes an FPGA with no defects. The ideal layout is applied to the arrangement of courtyards. If there are defective logic gates within a courtyard as well as unused gates which are not defective, then those unused gates replace the defective ones in the courtyard. If a courtyard contains enough spare gates to replace all its defective gates, it is a repairable courtyard. If all the courtyards on the chip are either defect free or repairable, the chip is deemed usable.

A variation of this technique sets aside a number of spare courtyards which are only used to replace a courtyard that isn’t repairable. Given enough spare courtyards to replace the non-repairable courtyards, the chip is usable.

A second method of improving yield is to use redundant rows. In this scheme, a number of spare rows of logic blocks are reserved. If a logic block is determined to be faulty, the entire row containing the block is shifted one row toward the spare. Each row between the defective row and the spare rows is also shifted one row to maintain the overall architecture. This scheme requires specialized selector circuits to select the rows to use. The defective row is disabled in the final implementation of the architecture [Hat93].

This technique can be easily adapted to use spare columns instead of rows. In addition, dividing the chip into blocks allows the use of multiple spare rows because

each block can contain a spare row for use within that block thus increasing the overall fault tolerance.

A major disadvantage of using spare rows (or columns) is the efficiency of the scheme. A spare row replaces an entire row in the FPGA whether the row has a single fault or the entire row is faulty. If only one block in the row is faulty, the rest of the row contains functional logic blocks that go unused. This scheme also reduces the level of fault tolerance achievable because once all the spare rows are used, there is no additional capacity available.

A variation of the redundant approach is to associate each CLB with a switch matrix [KI94]. At the end of each row of CLB/Matrix pairs, an extra pair is included. If a defect is found in either the CLB or the switch matrix, the pair is declared defective. The circuit is rerouted around the defective pair using bypass, connect left, connect right, or disconnect routing resources assigned to each switch matrix.

Another method is called node-covering [HD98]. Each node or cell in the FPGA has an associated cover cell which can be reconfigured to replace the node should it become defective. Nodes in the array that are actively used in the circuit are called primary nodes or cells. Primary cells cover other primary cells in a chained manner. The last cell in a row (or column) is a spare and covers the last primary cell in the row (or column). When a faulty cell is identified, it is replaced by its cover cell, which is replaced by its own cover cell. This continues down the chain until the spare cell is reached. Each cover cell must be able to produce the functionality of the cell it is covering. In an FPGA, this is easily done because all cells are the same. A cover cell must also be able to reproduce the routing of the cell it is covering with respect to the rest of the array. This requires cover segments in the routing. Each cover cell must include reserved segments associated with the cell it is covering so when the circuit is reconfigured, the covering cell can implement the correct routing.

A similar method matches defective logic blocks to a set of unused logic blocks in the circuit using a mini-max grid matching algorithm [EB97]. Once a complete set

of matching logic blocks is found that correctly implements the circuit functionality and minimizes the distance between matched blocks, a shifting strategy updates the circuit. Thus, a faulty block isn't necessarily swapped with its matching unused block. Instead, adjacent blocks between the matching blocks are shifted toward the unused block to minimize the length of interconnections.

Timing driven schemes minimize increasing operation circuit delays due to reconfiguration of the circuit. One method uses distributed edge slack to provide the best reconfiguration of the circuit using available resources [ML96]. Edge slack is the amount the route delay can increase without violating circuit timing constraints. This implies that a CLB can only be moved to certain other CLBs within its neighborhood. A slack neighborhood graph is constructed which identifies the neighborhood satisfying the edge slack constraint for each CLB. The circuit is reconfigured and the resulting circuit has the minimum circuit timing degradation while avoiding faulty logic blocks.

2.4.2 Reconfiguration. There are several reconfiguration techniques to deal with faults. One strategy incorporates fault detection within the circuit itself to indicate the presence and location of a fault at some level of granularity. Once a fault is detected, the replacement scheme is invoked which reroutes logic and connections to avoid the defective area. The faulty block is swapped out with a spare block. An alternative is to swap the entire row or column containing the faulty block [DP94, GASF03].

Another method partitions the circuit into a set of tiles. Each tile contains the logic and interconnect needed to perform a desired logical function plus some amount of unused resources. Multiple configurations of each tile are determined which do not use certain resources. In this way, if one of the resources within the tile becomes defective, a different configuration of the tile which implements the same logic function can be used. The interface of a tile to the rest of the circuit is defined and fixed so all configurations of that tile have the same interface. This permits swapping of

tiles without the overhead of reserved routing and interconnect [LMSP98a, LMSP98b, LMSP99, LMSP00, Els03].

Network flow techniques dynamically determine the optimal reconfiguration path for the FPGA when a fault occurs. Cells in the FPGA correspond to vertices on a network flow graph. Each vertex has two unidirectional edges to each neighboring cell forming North-South and East-West cell pairs. The algorithm finds a maximal set of reconfiguration paths in the design by determining a maximal flow in the FPGA flow graph [MD99].

A modification to the existing architecture of the switch blocks in the FPGA along with the addition of spare tracks eliminates the need to calculate all possible circuit configurations required by other methods. The switch blocks connecting the CLBs route the interconnect segments as needed for the circuit. Each segment entering a switch block can connect to another segment in one of three ways: a) the segment goes straight through, b) the segment connects to a segment going north, or c) the segment connects to a segment going south. The addition of spare interconnect segments between the switch blocks as well as the switches to accommodate them means a faulty segment can be replaced by a spare simply by changing the switch configuration inside the switch block. This arrangement can also reroute to avoid a faulty CLB [XSHL99].

The replacement schemes discussed so far require off-chip processing to replace faulty blocks and reroute all the interconnections so the circuit functions properly after reconfiguration. An alternative is to place the routing and reconfiguration in the blocks themselves.

The use of embryonic arrays is one such technique. This method uses a biologically-inspired algorithm for fault tolerance. A homogenous array of cells is formed from the logic blocks of the FPGA. Each cell contains processing and control elements and the configuration of every cell in the array. The coordinates of the cell in the array determine its function which is found in a lookup table inside the cell. In this way,

every cell is capable of performing the function of every other cell. Faulty cells are “killed” by eliminating the column containing the cell. The column becomes transparent to the rest of the array and a spare column takes its place. The coordinates of the replacement cells are changed to correspond to the dead cells. The new cell function is determined from the coordinates and the cell reconfigures itself [CT02, MGM⁺96].

Although the two approaches to fault tolerance have different goals, the methods are complimentary. In some cases, the methods used for yield enhancement can be just as applicable to dynamic fault tolerance. The biggest difference is the approach taken to diagnose and repair faults. Yield enhancement techniques usually involve some type of external testing and a permanent rerouting of the circuit on the chip so the defect tolerance is transparent to the user. Dynamic reconfiguration is more flexible and is implemented by the user. It provides a way for an operational system to continue functioning in spite of faults present in the hardware.

2.4.3 Performance of Reconfiguration Schemes. Performance of reconfiguration schemes is usually measured by the overhead incurred by the scheme. Provisioning with spare resources means increased overhead as a percentage of all available resources. The number of spares determines the level of fault tolerance. Ideally, an array with n spares will tolerate n faults. However, interconnection and routing within the FPGA limits the number of faults that can be tolerated [OT97].

Timing degradation is another performance measure for dynamically reconfigured systems [HTA94, CC95]. When logic blocks are moved or replaced, the interconnect routing between the blocks must be changed accordingly. This could cause an increase in the physical length of interconnect between blocks resulting in timing degradation. A circuit sensitive to timing differences would not benefit from a reconfiguration method which induces large timing degradation.

A third, and possibly most important, performance metric is the reconfiguration time. As stated earlier, in terms of today’s computing speeds, reconfiguration of an FPGA is relatively slow. A reconfiguration scheme with a short configuration time

will generally be preferable to a slower scheme. This is certainly true for critical systems that need to minimize downtime.

2.5 Summary

Fault tolerant computing has its roots in vacuum tube computers developed in the 1940s. The techniques used by engineers to overcome defective components enabled their computers to continue to function.

A distinction is made between failures, faults, and errors though they are closely related. Fault tolerance is one way to improve system reliability, but it is not a requirement for reliability. A key ingredient for achieving fault tolerance is redundancy of information, time, hardware, or software.

FPGAs provide a convenient source of redundant hardware. FPGAs are constructed of a regular array of CLBs which can be programmed to implement a combinational or sequential logic circuit. Redundancy exists in the form of unused CLBs in the array.

Several techniques have been proposed to take advantage of an FPGA's inherent redundancy and its reconfigurability. Most techniques replace faulty CLBs with spare ones through reconfiguration of the circuit.

III. Research Methodology

3.1 Problem Definition

3.1.1 Goals and Hypothesis. The ability of an FPGA to reconfigure makes it an effective device for implementing fault tolerance through circuit reconfiguration. Fault tolerance at this level detects faults while the user's circuit is operating and then replaces the faulty components of the circuit.

The goals of this research are to implement a circuit reconfiguration system that uses one of four replacement methods and to determine which of the methods has the shortest reconfiguration time. It is expected reconfiguration time will be affected by the replacement method used. A replacement method which takes advantage of the inherent column architecture of the FPGA will likely have a lower reconfiguration time due to the basic design and inter-operability of the components in the FPGA. However, the direction of the replacement is not expected to have a significant effect on the reconfiguration time. It is also expected that circuit congestion after a CLB is moved will significantly impact the reconfiguration time with a congested circuit having a longer reconfiguration time than a non-congested circuit.

Goal: Implement a CRS that uses row replacement and column replacement to reconfigure an FPGA and measure the reconfiguration times for moving a row up and down and a column left and right to determine which replacement method performs best.

Hypotheses: Reconfiguration that uses the inherent features of the FPGA's architecture will likely have a lower reconfiguration time. Direction of the replacement (up/down or left/right) is not expected to affect reconfiguration time, but the relative circuit congestion is likely to have an impact on the reconfiguration time for a specific replacement method.

3.1.2 Approach. To achieve the goal of this study, a functioning circuit is implemented on an FPGA. The configuration bit file for the circuit is modified to

achieve a row-wise or column-wise replacement. The modified bit file is stored in memory until needed by the CRS.

Once a fault is detected, the CRS is notified and the reconfiguration process begins. The modified bit file reconfigures the FPGA according to the replacement method. The reconfiguration time is measured from the moment a fault indication is received at the CRS until FPGA reconfiguration is complete.

Four replacement methods are examined; row up, row down, column left and column right. Upon notification of a fault, the row replacement method moves the circuit up or down one row during the reconfiguration process of the FPGA. The column replacement method functions in the same way except the circuit is moved over one column. After reconfiguration, proper operation of the circuit is verified.

Detection of faults and defects in the FPGA fabric is provided by an external system not specified here. It is assumed a fault has been detected (by some method) and the location of the fault is known.

3.2 System Boundaries

The CRS is the system under test (SUT) and is shown in Figure 3.1. The system consists of the FPGA and its various resources including the configuration memory, CLBs, and routing resources, and the test circuit implemented in the FPGA.

The component under test (CUT) is the replacement method. The four methods under study are column left, column right, row up and row down replacement. There are other methods of replacement including individual CLB replacement and shifting that are not considered.

Numerous FPGA architectures could be used with the CRS. However, the Xilinx Spartan 3 FPGA is used in this study.

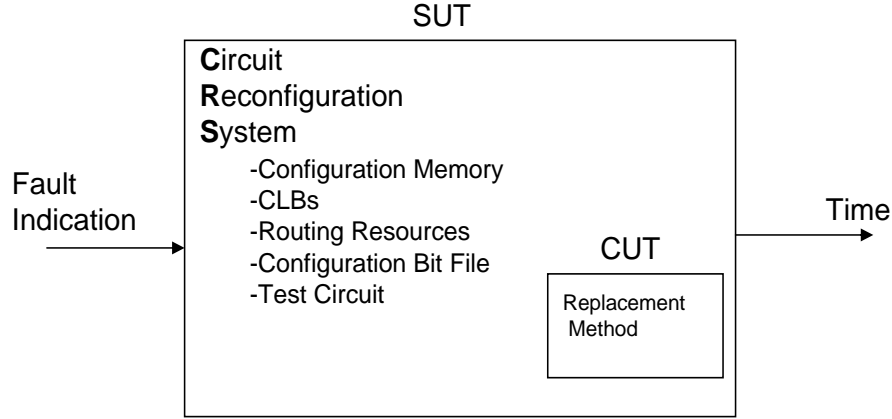


Figure 3.1: The system boundaries of the Circuit Reconfiguration System

3.3 *System Services*

The service provided by the CRS is the reconfiguration of the FPGA. The outcome of the service is the implementation of a circuit on the FPGA. A positive outcome occurs when the circuit continues to function as required. Conversely, a negative outcome occurs when the circuit doesn't function as intended. A failure may be due to additional faults in which case the CRS will receive another fault notification and start the reconfiguration process again. Another type of failure occurs when the modified bit file cannot be implemented in the FPGA due to a lack of routing resources or unused CLBs. Finally, a failure occurs if the FPGA has so many faults that there aren't enough CLBs or routes to implement the circuit.

3.4 *Workload*

The workload submitted to the system is simply a fault indication input to the CRS. One fault at a time may be applied to the system. Another fault may not be applied until the reconfiguration cycle initiated by the preceding fault is complete.

3.5 Performance Metrics

An important consideration when using FPGAs is the configuration time. The time required to reconfigure a circuit is time the circuit is out of operation. Obviously, it is desirable that configuration time be as short as possible.

The primary metric for this study is reconfiguration time. To evaluate the effectiveness of the four replacement methods, the time to complete a reconfiguration cycle using each method is measured.

3.6 Parameters

3.6.1 System.

- The replacement method is the chief system parameter. The replacement method determines how the circuit is moved within the FPGA.
- The FPGA used in the system is another system parameter. The arrangement of CLBs and routing resources in the FPGA plays a role in how effective the replacement method is in terms of reconfiguration time and compatibility. For some architectures, a row replacement method may not be possible. Conversely, column replacement may not be well suited to other architectures.
- The length of the configuration bit stream is determined by the number of CLBs in the FPGA. The longer the bit stream, the more CLBs and routing that need to be configured and the longer the configuration time.
- The test circuit implemented in the FPGA can be nearly any circuit with a range of complexities, capabilities, and sizes. For an initial configuration, the length of the bit file and the configuration time are independent of the test circuit. However, partial reconfiguration uses a difference bit file created by comparing the original and modified bit files. The difference bit file is smaller than a bit file used for a full configuration. Partial reconfiguration only configures the CLBs necessary to implement the modifications resulting in a lower configuration time compared to a full configuration.

- The relative congestion in the FPGA of the area to which a CLB is moved is determined by, among other things, the size of the circuit implemented on the FPGA and the number of unused CLBs. For this study, a clear area is defined as one which has no occupied CLB within two rows or columns of the relocated CLB while a congested area has at least one occupied CLB adjacent to the relocated CLB.

3.6.2 Workload. The rate at which faults are input to the system is the sole workload parameter. The input fault rate cannot be greater than the inverse of the configuration time of the FPGA. Faults applied to the CRS at a rate greater than the maximum will not cause another reconfiguration until the previous configuration cycle is complete.

3.7 Factors

Since the primary goal of this research is to measure the reconfiguration time using a specific replacement method, the factors for this study are the replacement method used by the CRS and the circuit congestion (or location type) near the relocated CLB. The factor levels are shown in Table 3.1. It is expected that in the Spartan 3 FPGA the column replacement method will be faster than the row replacement method because of the layout of the CLBs. However, it is not expected that there will be a significant difference in the reconfiguration time of column left versus column right and row up versus row down. The circuit congestion will likely impact the reconfiguration time with a congested area having a higher reconfiguration time than a clear area.

Table 3.1: Factors and levels for the CRS

Column Left		Column Right		Row Up		Row Down	
Clear	Congested	Clear	Congested	Clear	Congested	Clear	Congested

3.8 Evaluation Technique

There are no analytical or simulation models available to evaluate this system. Thus, the evaluation technique used for this study is direct measurement. Upon initialization, a functioning circuit is placed in a fault free FPGA. The functioning circuit is a 4-input LUT implemented in one CLB. A simulated fault is detected and the CRS is notified of the fault and its location in the FPGA. The CRS causes a new configuration bit file to be loaded onto the FPGA. The bit file is a modified version of the bit file used to program the initial circuit. The modification causes the CLB to be moved one column or one row. Reconfiguration of the FPGA is done through the Joint Test Action Group (JTAG) port.

A timer circuit measures the reconfiguration time. The timer circuit is on a separate FPGA so as to not be affected by the reconfiguration process. The timer measures the number of system clock cycles the test clock (TCK) is active. The TCK signal clocks the configuration bit stream through the JTAG port.

Validation of the measured results is accomplished using the expected configuration time through JTAG for the given FPGA. An estimate of the configuration time can be found by dividing the size of the configuration bit file (in bits) by the TCK frequency. This estimate is adjusted as necessary to account for differences in the definition of the start and stop points of the configuration cycle.

3.9 Experimental Design

For this study, a full factorial design is used. The number of replications is based on the expected variance of the data. It is expected that the measured reconfiguration times will have a small variance. Ten replications is expected to be sufficient to perform the statistical analysis. Since there are two factors with two and four levels respectively, a full factorial design requires eight experiments. In addition, for each level of circuit congestion, three different locations in the FPGA are chosen to perform the measurements. For convenience, the clear locations are named A1, A2, and A3.

The congested locations are named B1, B2, and B3. Table 3.2 is a cross reference of the location names to the slices in the FPGA that correspond to the location. The three slices listed plus one unused slice comprise the CLB at a particular location. A total of 24 experiments are conducted. The resolution of the measurements is determined by the system clock used for the timer. The smallest increment between measured values is one clock period. For a more detailed description of the experimental setup, see Appendix A.

Table 3.2: Location names and the associated slice numbers in the FPGA corresponding to the location.

A1	A2	A3	B1	B2	B3
X10Y30	X10Y12	X28Y40	X30Y18	X24Y20	X30Y14
X10Y31	X10Y13	X28Y41	X30Y19	X24Y21	X30Y15
X11Y30	X11Y12	X29Y40	X31Y18	X25Y20	X31Y14

3.10 Results Analysis

The 2-factor analysis of variance (ANOVA) method is used to determine the statistical significance of the main and interaction effects with respect to the errors. The effects, predicted responses, and ANOVA are at the 90% confidence level.

The analysis assumes the errors are statistically independent, normally distributed, and have a constant standard deviation. The independence and homoscedasticity of the errors is verified by examination of the plot of predicted response versus error. The assumption of normally distributed errors is verified using a normal probability plot.

Since the reconfiguration time is a lower better metric, the best performing replacement method and location type is the one with the lowest reconfiguration time.

3.11 Summary

This research determines the performance of the circuit reconfiguration system. The system parameters of reconfiguration method and circuit congestion are used in a full factorial experiment which measures the reconfiguration time of a Spartan 3 FPGA. The data is analyzed using the ANOVA technique to determine the best method.

IV. Data Analysis

4.1 Introduction

The goals of this research are to implement a CRS that uses row and column replacement methods to reconfigure an FPGA and determine which of the four replacement methods has the shortest reconfiguration time. It is believed that, given the columnar design of the Spartan 3 FPGA architecture, a column replacement method will have a shorter reconfiguration time than a row replacement method. It is also speculated that the direction of the replacement will not affect the reconfiguration time. Thus, the reconfiguration time for a column left replacement would be the same as the time for a column right replacement. The same could be said for the row up and row down methods. Finally, it is believed the relative circuit congestion after a row or column movement would impact reconfiguration time with a clear area having a lower time than a congested area.

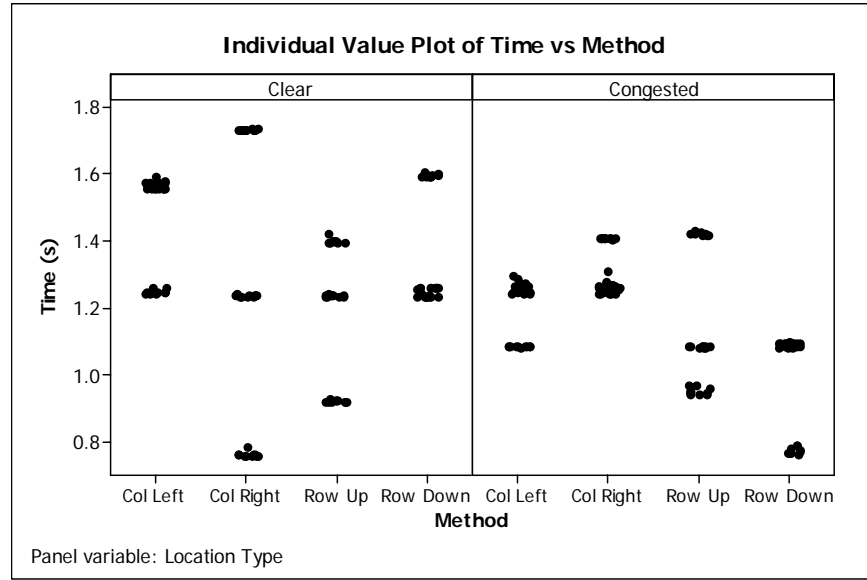


Figure 4.1: Measured reconfiguration time by replacement method and location type.

4.2 Validation

A plot of the raw data values is shown in Figure 4.1. There is a wide range of values in the reconfiguration times. The groupings of the values for a particular

method are attributed to the three locations within the location type used during the experiments. The magnitude of the reconfiguration time is not the focus of this study. Rather, it is to determine which replacement method is fastest.

The reconfiguration time is defined to be the length of time the JTAG TCK clock is active. The TCK signal is used by the JTAG port to clock the configuration bits into the FPGA. Thus, the size of the configuration bit file (in bits) divided by the frequency of the TCK clock will give an estimate of the configuration time. The clock rate of the TCK signal used in this study is 200 KHz [XAR03]. The average file size of the configuration bit files is 230,682 bits. Thus, it is expected that the measured values will be approximately 1.15 seconds. Reviewing the measured values shown in Figure 4.1, the data is within a reasonable range of this estimate and is therefore considered valid.

4.3 Initial Analysis

The initial analysis of the reconfiguration time versus the location type (clear or congested) and the replacement method (column left, column right, row up, and row down) treated the three locations for the clear location type as one location. Similarly, the three locations for the congested location type were combined. Thus, for each combination of location type and replacement method, there are 30 data points.

4.3.1 Analysis by Location Type and Method. A two-factor analysis of variance was conducted to determine the main and interactive effects. From Figure 4.2, it appears the errors are independent and have a constant standard deviation. However, Figure 4.3 does not indicate they are normally distributed. Since this violates the normality assumption necessary for an ANOVA, the results of the ANOVA cannot be used to draw any conclusions.

Figure 4.4 shows the 90% CIs for the mean reconfiguration times separated by location type and replacement method. The Tukey method for multiple comparisons

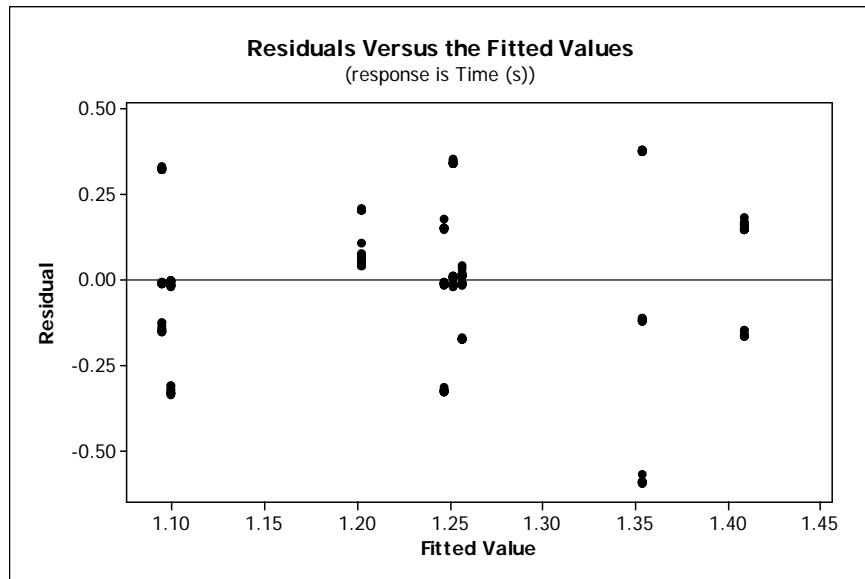


Figure 4.2: Residuals versus fitted values for locations combined by type.

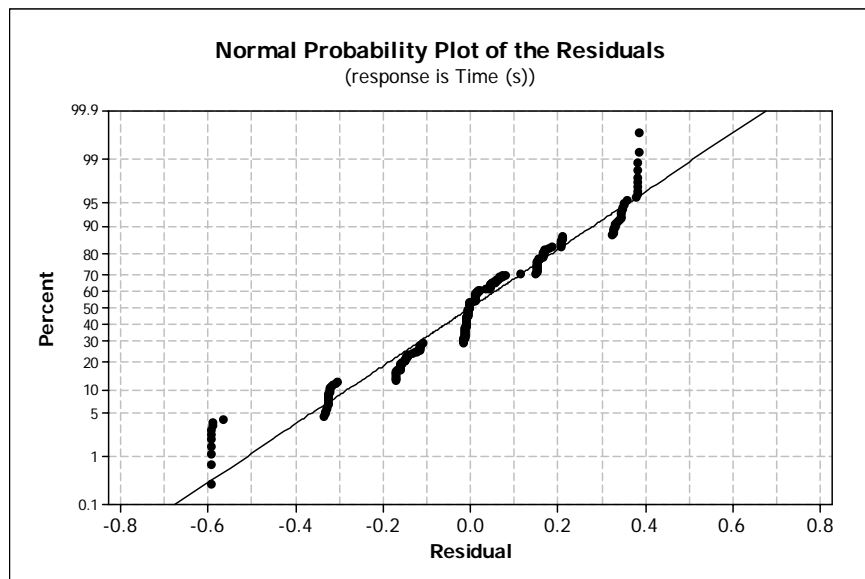


Figure 4.3: Normal probability of the residuals for locations combined by type.

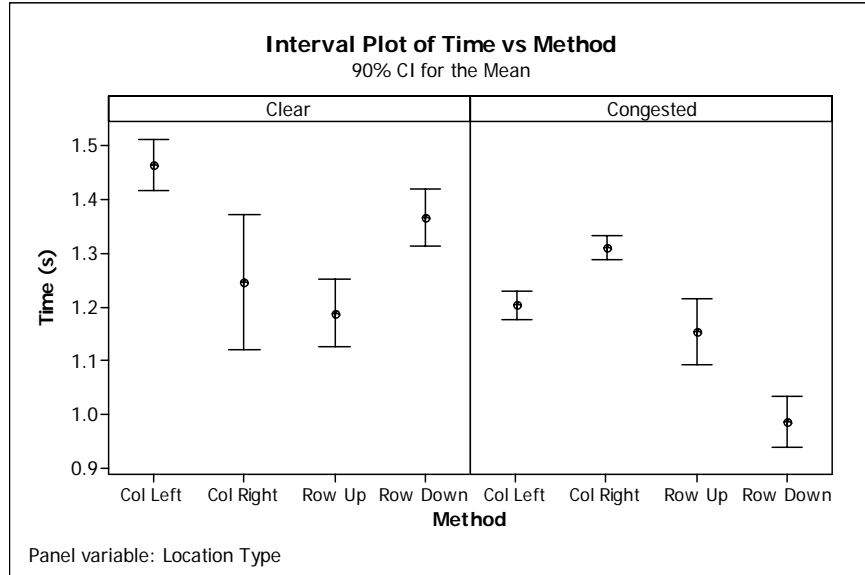


Figure 4.4: 90% confidence intervals on the mean of reconfiguration time for clear and congested areas by replacement method.

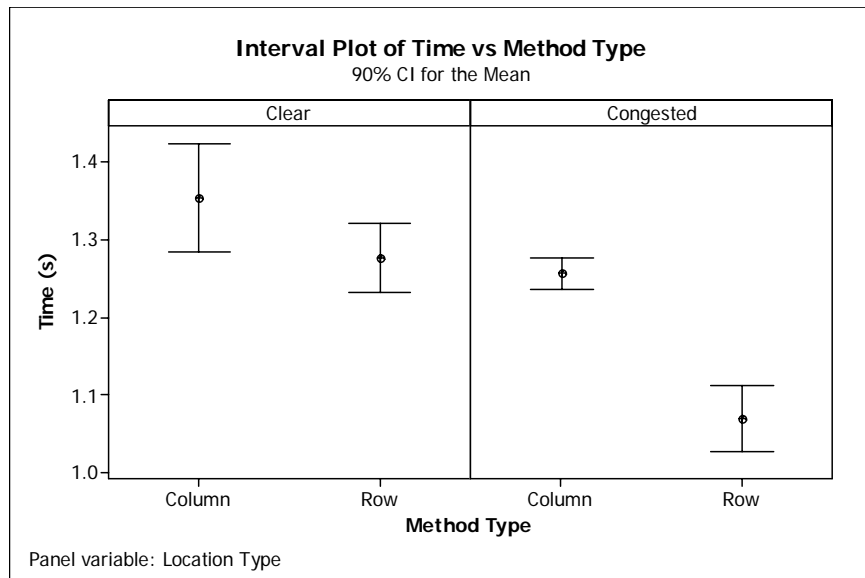


Figure 4.5: 90% confidence intervals on the mean of reconfiguration time for clear and congested areas by replacement method type.

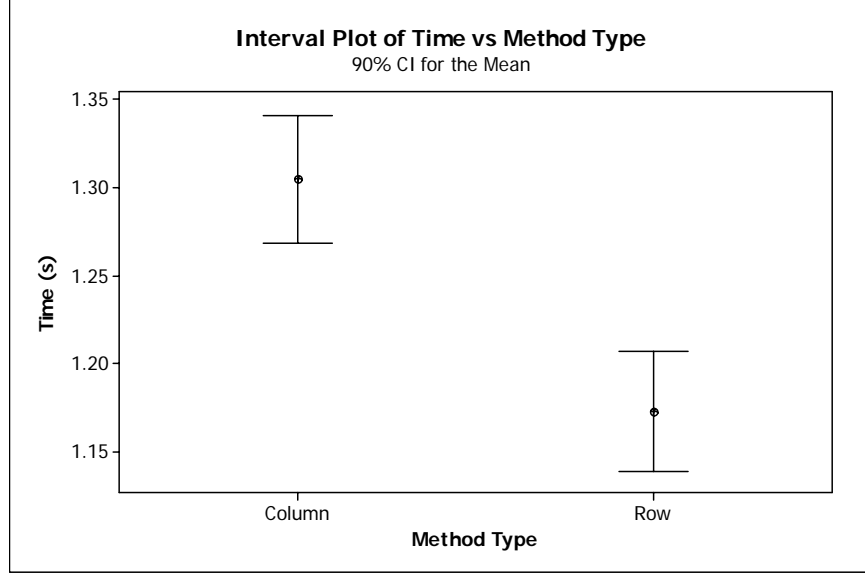


Figure 4.6: 90% confidence intervals on the mean of reconfiguration time by replacement method for both location types combined.

was used to do pair-wise, simultaneous comparisons of the CIs of reconfiguration times for each location type. From the comparison results, the following observations at the 90% confidence level are made with respect to the reconfiguration times:

- For a **clear** location type, there is no statistical difference between the column right and row up methods or the column right and row down methods.
- For a **clear** location type, there is a statistical difference between the row up and row down methods.
- For a **clear** location type, there is a statistical difference between the column left and column right methods and the column left and row up methods.
- For a **clear** location type, the column left and row down methods are not statistically different.
- Based on these observations for a **clear** location type, it is not possible to determine which method is fastest or slowest.
- For a **congested** location type, the column left, column right, and row down methods are statistically different.

- For a **congested** location type, the row up method is statistically different from the row down and column right methods.
- There is no statistical difference between the column left and row up methods for a **congested** location type.
- For a **congested** location type, the column right method is the slowest replacement method while the row down method is the fastest.

Figure 4.5 shows the 90% CIs for the two column methods combined and the two row methods combined while Figure 4.6 displays the 90% CIs for the two replacement methods with both location types combined. From Figure 4.5, it can be observed that for both clear and congested areas, the combined row methods are faster than the column methods. Figure 4.6 indicates that when both location types are combined, the row methods are faster than the column methods.

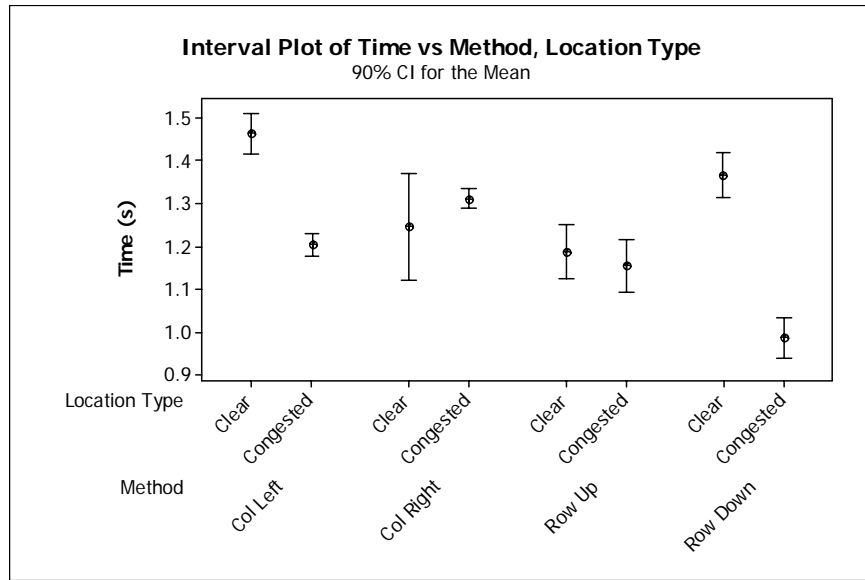


Figure 4.7: 90% confidence intervals on the mean of reconfiguration time by replacement method and location type.

Another way to view this data is shown in Figure 4.7 which groups the 90% CIs for each location type by a particular replacement method. For the column left and row down methods, the CIs by location type do not overlap at all meaning clear and congested locations have statistically different reconfiguration times for these two

methods. The CIs by location type for the column right and row up methods overlap with the mean of one included in the CI of the other indicating that reconfiguration times for clear and congested locations are not statistically different.

Using the Tukey method for multiple comparisons of CIs it can be determined with 90% confidence that the row down replacement method in a congested location has the fastest reconfiguration time. In addition, since the column left and row down replacement methods in a clear location are not statistically different, it isn't possible to determine which combination of method and location type has the slowest reconfiguration time.

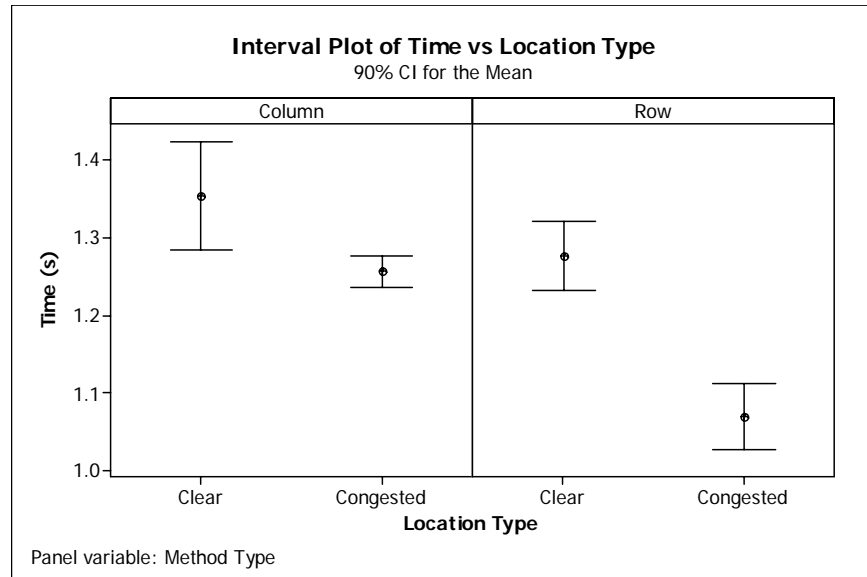


Figure 4.8: 90% confidence intervals on the mean of reconfiguration time by location type for each method type.

Figure 4.8 shows the 90% CIs when the two column methods and the two row methods are grouped by method type while Figure 4.9 shows the 90% CIs of the reconfiguration times for the two location types for all replacement methods combined. Both figures indicate that reconfiguration in a congested area is faster than reconfiguration in a clear area.

4.3.2 Comparison of Observations with Hypotheses. A review of these observations leads to several inconsistencies with respect to the previously stated hypothe-

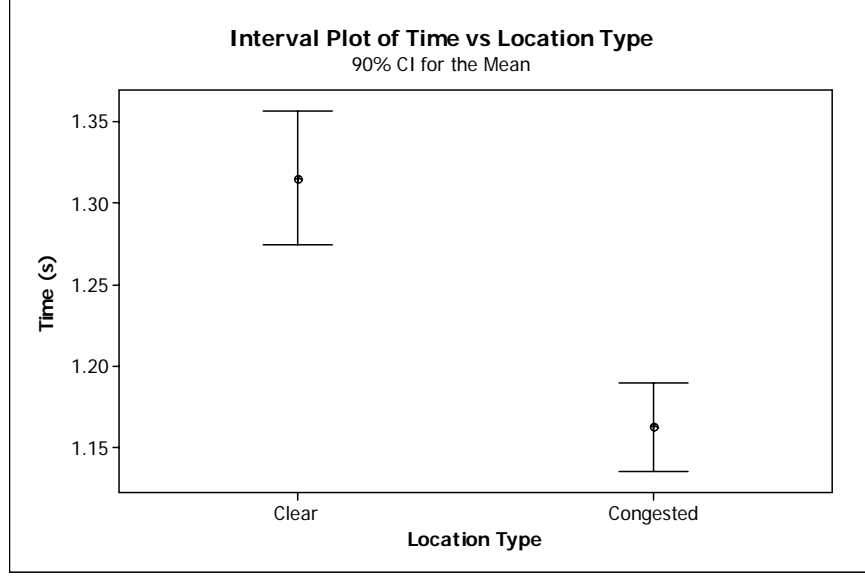


Figure 4.9: 90% confidence intervals on the mean of reconfiguration time by location type for all methods combined.

ses. First, when the four replacement methods are considered separately (Figure 4.4), it clearly is not the case that the column replacement methods are faster than the row replacement methods for a congested location and no conclusion can be drawn for a clear location. When two replacement methods are grouped by method type (Figure 4.5), the row replacement methods are faster for both clear and congested areas. Additionally, when the two location types are grouped (Figure 4.6), the row methods are faster than the column methods.

Second, the direction of the replacement was statistically significant in all cases (Figure 4.4), but interestingly, the faster directions for the clear locations (right and up) are opposite the faster directions for the congested locations (left and down).

Finally, the data indicates that reconfiguration in a congested area is actually faster than in a clear area when the replacement methods are grouped by type (Figure 4.8). The same is true when all replacement methods are combined (Figure 4.9). When the four replacement methods are considered separately (Figure 4.7), reconfiguration in a congested area is faster for the column left and row down methods. The overlap of CIs for the column right and row up methods prevents a determination of

which location type has faster reconfiguration times. Thus, it cannot be determined if the location type affects the reconfiguration time for either of these two methods.

4.4 *Second Analysis*

The inconsistency with which the data can be used to clearly portray one replacement method as the best in terms of reconfiguration time could be due to the grouping of the three locations into one location type. Based on the invalid ANOVA assumptions discussed in the first analysis and these inconsistencies, a second analysis splits the data into two sets by the location type. The data is further separated by the specific location yielding three locations for each type with ten data points at each location. Separate ANOVAs were conducted for each location type with the factors being the replacement method and the specific location.

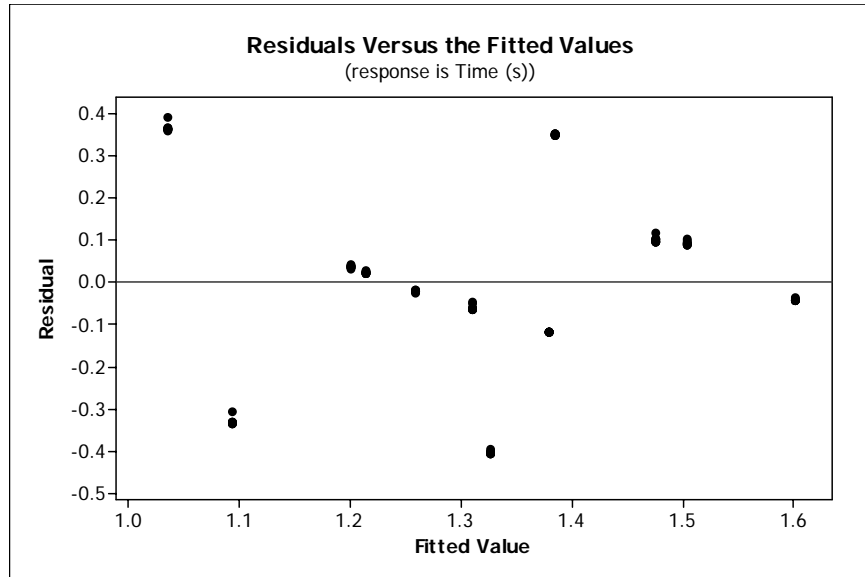


Figure 4.10: Residuals versus fitted values for clear locations.

4.4.1 Analysis of Clear Locations. From Figure 4.10, the independence and homoscedasticity of the residuals is verified for the clear locations. However, as was the case with the first analysis, errors are not normally distributed, as shown in Figure 4.11. Once again, the assumption of normally distributed errors necessary to perform

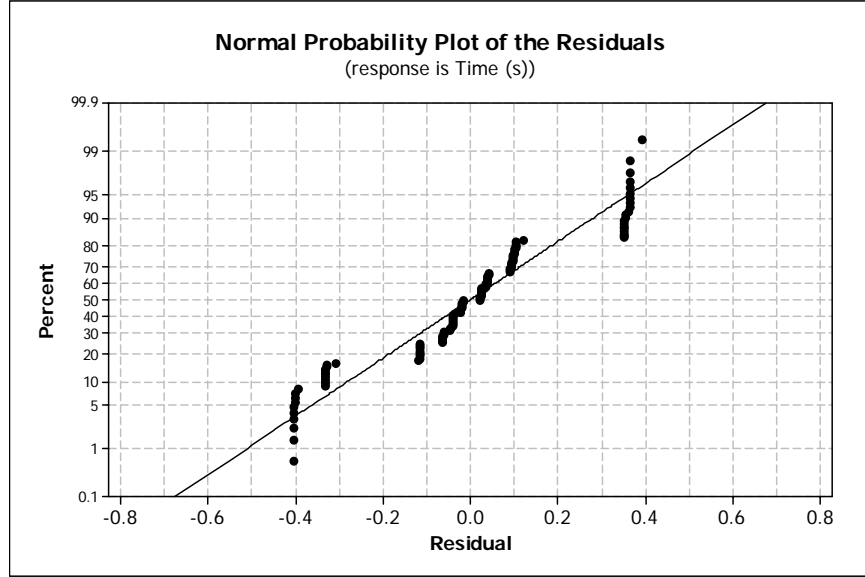


Figure 4.11: Normal probability of the residuals for clear locations.

an ANOVA is violated and the ANOVA results cannot be used. However, an analysis of the 90% confidence intervals on the reconfiguration time was conducted. As in the first analysis, the Tukey method for multiple comparisons was used where appropriate to do simultaneous comparisons of the CIs for each replacement method.

Figures 4.12, 4.13, 4.14, and 4.15 show the 90% confidence intervals for the mean reconfiguration times by replacement method for each clear location. The main observation from these plots is that for each replacement method, there is a statistical difference at the 90% confidence level between the reconfiguration times at the three specific locations. Another thing worth noting is the size of the confidence intervals. The CIs are very small indicating a very small variance in the measured values.

4.4.2 Analysis of Congested Locations. Figure 4.16 shows the residuals versus fitted values for the congested locations and Figure 4.17 shows the normal probability of the residuals. As before, the residuals appear independent and homoscedastic, but they are not normally distributed invalidating the ANOVA results. Figures 4.18, 4.19, 4.20, and 4.21 show the 90% confidence intervals for the mean reconfiguration times by replacement method for each congested location. After applying the Tukey

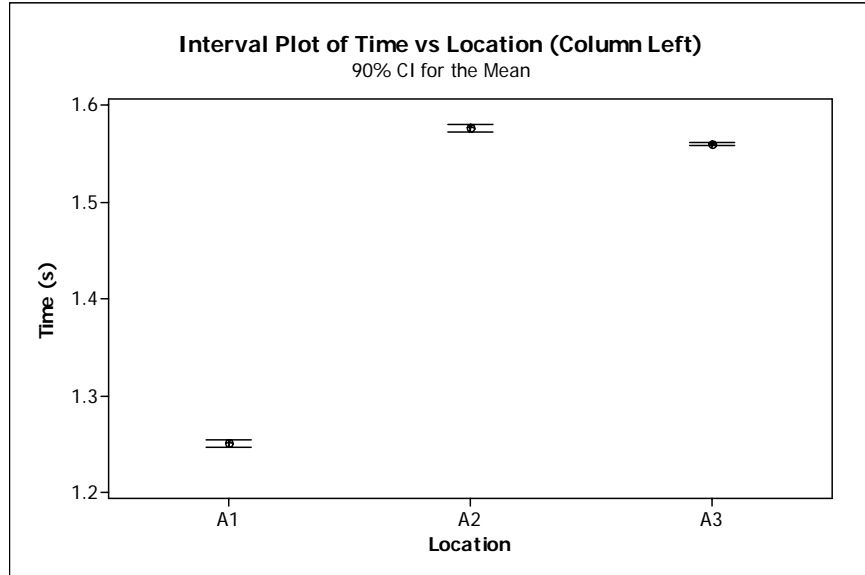


Figure 4.12: 90% confidence intervals on the mean of reconfiguration time for clear locations using the column left replacement method.

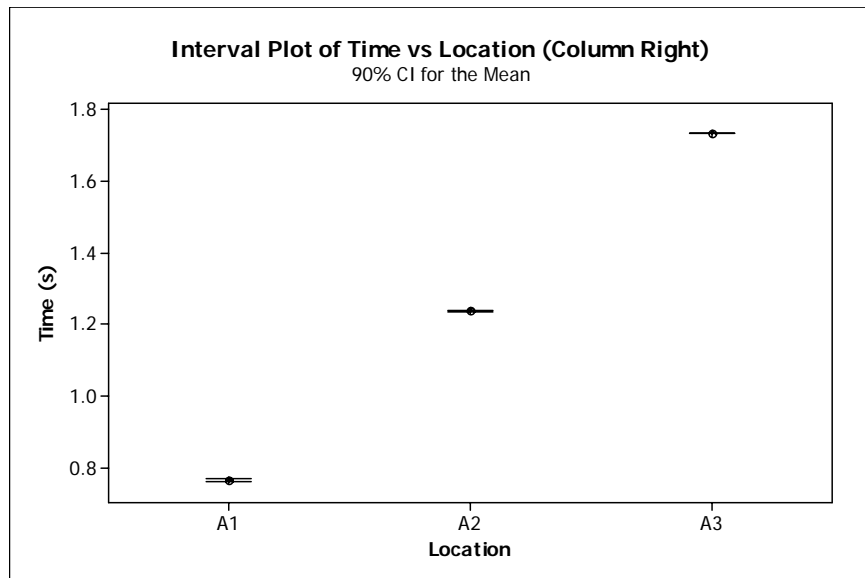


Figure 4.13: 90% confidence intervals on the mean of reconfiguration time for clear locations using the column right replacement method.

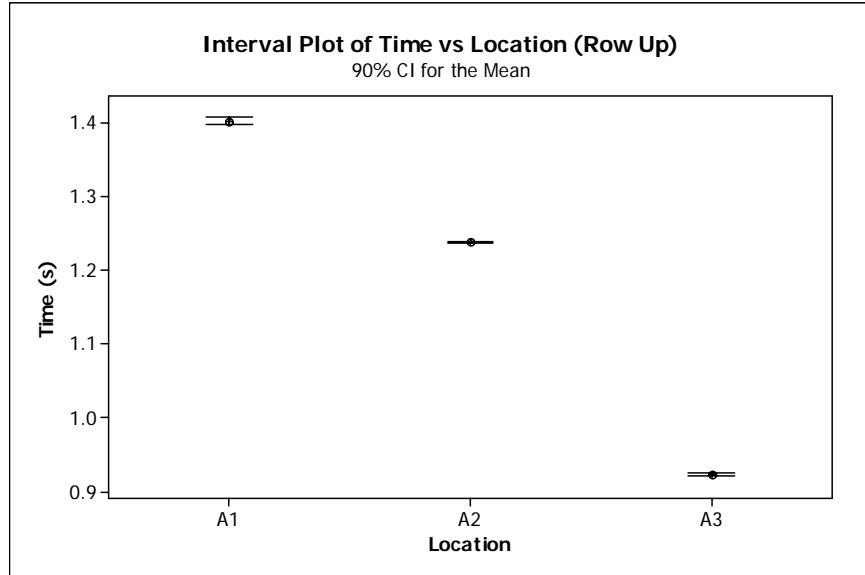


Figure 4.14: 90% confidence intervals on the mean of reconfiguration time for clear locations using the row up replacement method.



Figure 4.15: 90% confidence intervals on the mean of reconfiguration time for clear locations using the row down replacement method.

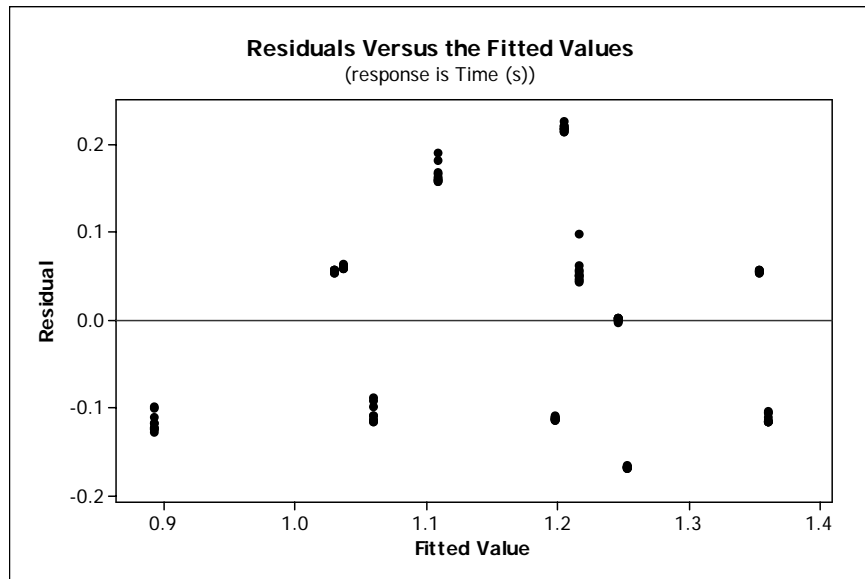


Figure 4.16: Residuals versus fitted values for congested locations.

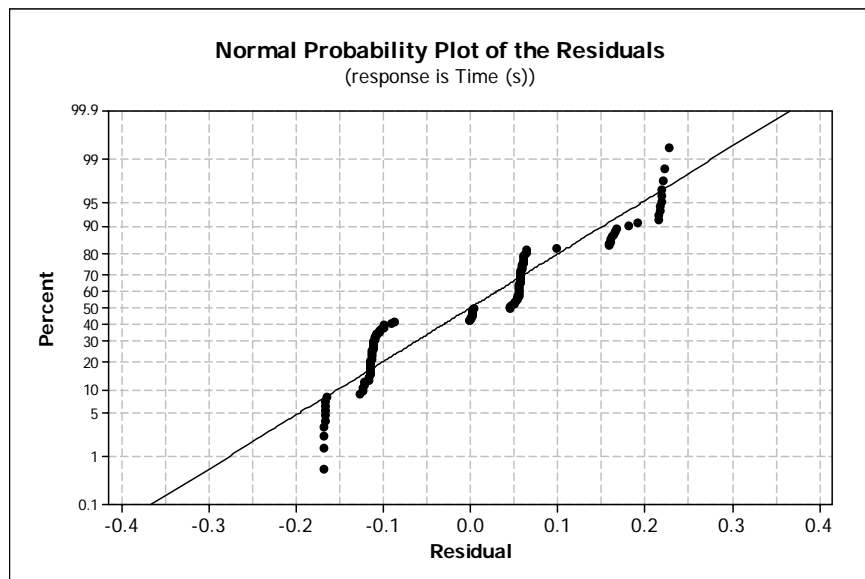


Figure 4.17: Normal probability of the residuals for congested locations.

method to compare the CIs for each replacement method, all measured reconfiguration times for a particular replacement method were found to be statistically different from each other.

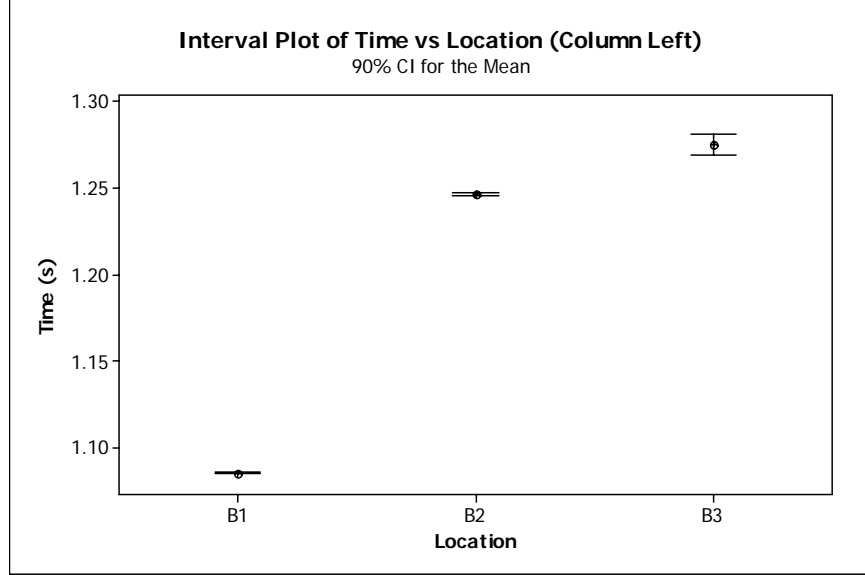


Figure 4.18: 90% confidence intervals on the mean of reconfiguration time for congested locations using the column left replacement method.

4.4.3 Analysis by Specific Location. The preceding two analyses of the reconfiguration times according to the location type indicate that grouping the three locations into one location type, as was done in the initial analysis, was incorrect. This grouping assumed that the specific location would not affect the reconfiguration time and that only the location type mattered. This is clearly not the case.

Figures 4.22 and 4.23 show the data grouped by location. The CIs for each replacement method at a particular location were compared using the Tukey method. From these pair-wise comparisons and the two figures, it was determined that there is no consistently fastest replacement method for a particular location type. For locations A1 and A3, the fastest method is column right and row up, respectively. However, for location A2, the column right and row up methods are not statistically different at the 90% confidence level. Similarly, the column left and row down methods are fastest for locations B1 and B3, respectively, while there is no statistical difference

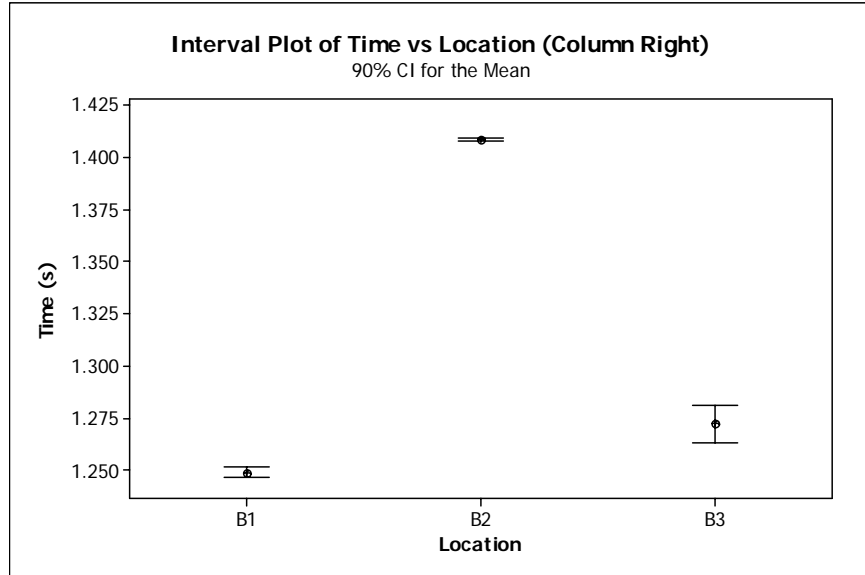


Figure 4.19: 90% confidence intervals on the mean of reconfiguration time for congested locations using the column right replacement method.

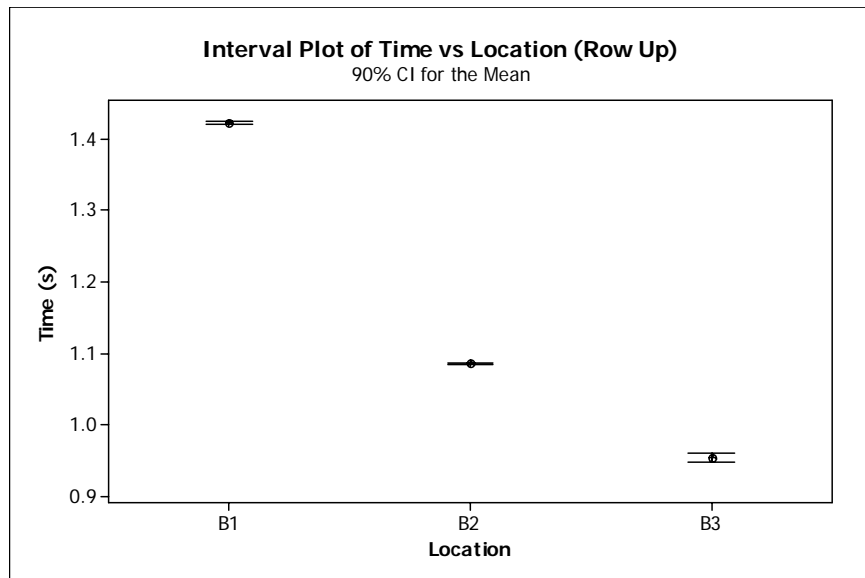


Figure 4.20: 90% confidence intervals on the mean of reconfiguration time for congested locations using the row up replacement method.



Figure 4.21: 90% confidence intervals on the mean of reconfiguration time for congested locations using the row down replacement method.

between the row up and row down methods at location B2 making it impossible to conclude which is faster.

Figures 4.24 and 4.25 show the data with the replacement methods combined by method type. As before, there is no consistently faster method type for a particular location type. However, at four of the six locations tested, the row method proved faster than the column method.

4.4.4 Comparison of Results of Second Analysis and Hypotheses. As was the case in the first analysis, some inconsistencies with respect to the hypotheses were noted while examining Figures 4.22 and 4.23. First, it was possible to determine the fastest replacement method only for locations A1, A3, B1, and B3. Interestingly, each of the four replacement methods was the fastest for one of these locations. The fastest method could not be determined for locations A2 and B2 except to say that at location B2, the fastest method was not a column method. When the replacement methods are grouped by type (Figures 4.24 and 4.25), the row method was faster than the column method for most locations, but it was not consistent among all locations.

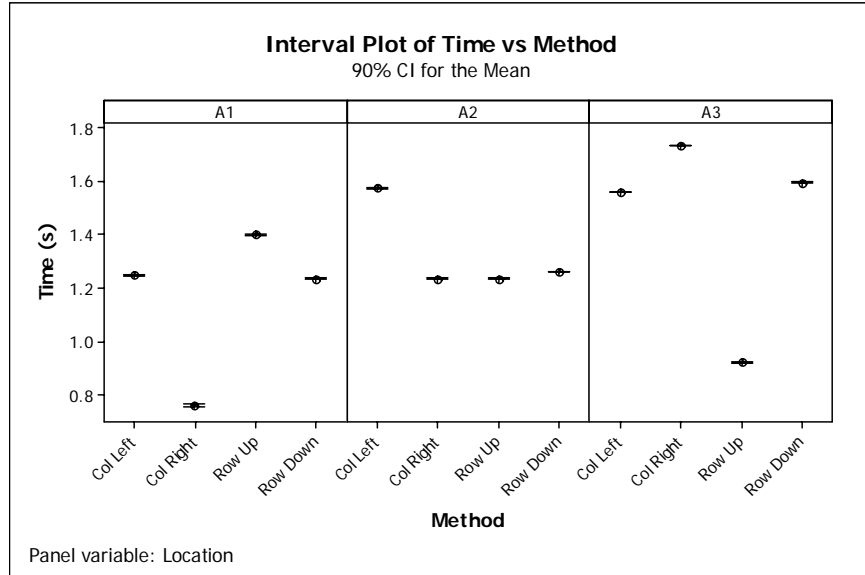


Figure 4.22: 90% confidence intervals on the mean of reconfiguration time by replacement method for clear locations.

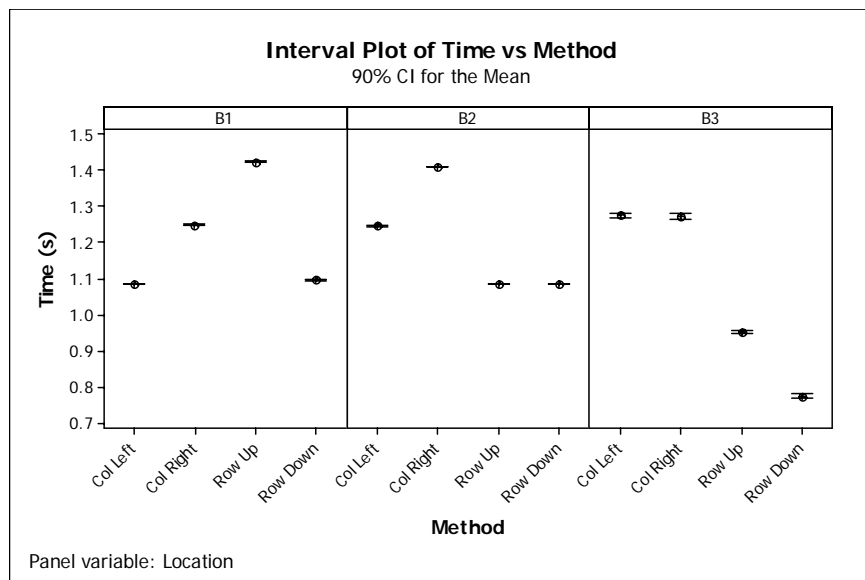


Figure 4.23: 90% confidence intervals on the mean of reconfiguration time by replacement method for congested locations.

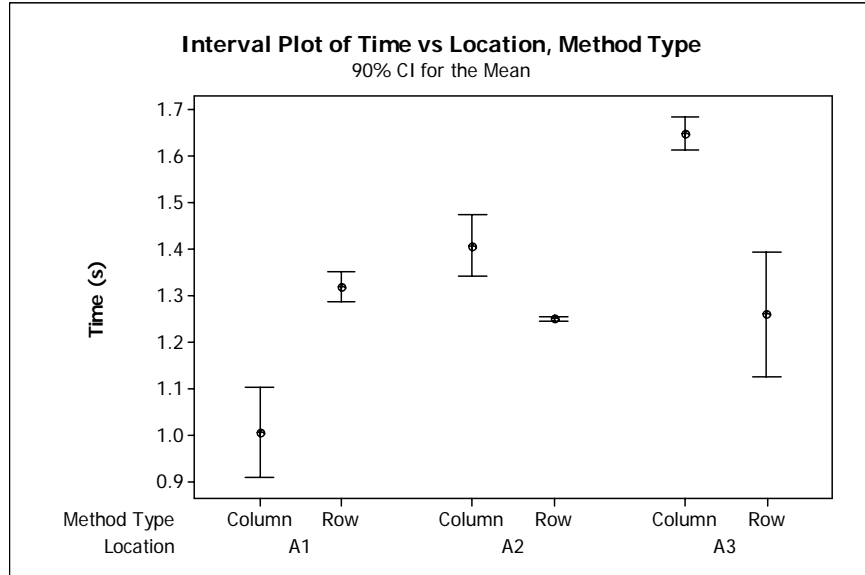


Figure 4.24: 90% confidence intervals on the mean of reconfiguration time by replacement method type for clear locations.

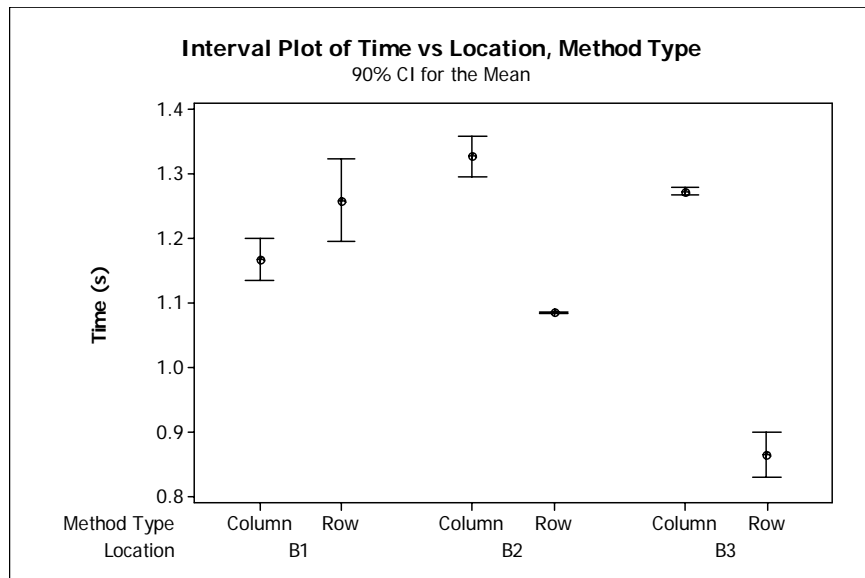


Figure 4.25: 90% confidence intervals on the mean of reconfiguration time by replacement method type for congested locations.

Second, comparing the reconfiguration times by direction at each location (Figures 4.22 and 4.23) shows that at location B2 the row up and row down methods are not statistically different and at location B3 the column left and column right methods are not statistically different. Similarly, at location A2, the row up and row down methods have no statistical difference at the 90% confidence level. At all other locations, there is considerable variation in the reconfiguration times according to the direction of the replacement.

The second hypothesis was that the direction of the replacement method would have no impact on the reconfiguration time. As shown in Figure 4.22 for the three clear locations, the reconfiguration times differ greatly between left and right for the column method. The same is true between up and down for the row method with the exception being location A2. However, even at A2, there is a statistical difference between row up and row down at the 90% confidence level. The same general observations can be made for the congested locations shown in Figure 4.23. There is a statistically significant difference between left and right and up and down with the exceptions being the row method at location B2 and the column method at location B3. In these two instances, the confidence intervals overlap and also encompass the means.

Since the data was split by location type for this analysis and there were no comparisons between clear and congested locations, the third hypothesis is rendered a moot point.

4.5 *Summary*

This chapter presented the measured data for the CRS. Two separate analyses were accomplished to determine the effects of the two factors. The first analysis lumped the three individual clear locations together into one location type. The same was done with the three congested locations. The data was then analyzed to show the effects of location type and the different replacement methods. The initial analysis showed that it was necessary to separate the locations and examine the data

according to the location type. In the second analysis, the effects of the specific location and the replacement method were determined within each location type. In the following chapter, the conclusions drawn from these analyses are presented along with suggestions for future research.

V. Conclusions

5.1 *Introduction*

This chapter presents the conclusions drawn from this research and the analysis presented in Chapter 4. The significance of this study as well as recommendations for future research are also discussed.

5.2 *Problem Summary*

One method of achieving fault tolerance in a system is to reconfigure the faulty circuit to overcome deficiencies. The reconfigurability of an FPGA makes it a logical choice for this fault tolerant approach. A circuit reconfiguration system can take a fault indication as an input and reconfigure the FPGA using one of four replacement methods: column left, column right, row up, or row down. To minimize system impact, it is necessary to determine which method is fastest for a given circuit environment to include the circuit congestion and the FPGA architecture.

5.3 *Conclusions of Research*

The first hypothesis of this study is that a column replacement method will have a faster reconfiguration time than a row replacement method due to the columnar architecture of the Spartan 3 FPGA. Based on the two analyses that were performed, this hypothesis cannot be affirmed. There were certain cases where column replacement was faster than row replacement, but this was not true in every case, with some cases being indeterminate.

The second hypothesis is that the direction of the replacement will have no impact on the reconfiguration time. As with the first hypothesis, this could not be determined consistently for all locations and location types. In fact, in most instances, there was a statistically significant difference in reconfiguration times based on direction. Thus, this hypothesis cannot be proven.

The last hypothesis is that a congested area will have a higher reconfiguration time than a clear area. This also proved to be false as no consistent determination

could be made for all four replacement methods. The column left and row down methods contradicted the hypothesis while the column right and row up methods were indeterminate. When the replacement methods were combined by method type, the congested areas had lower reconfiguration times. Congested areas also had lower reconfiguration times when all replacement methods were combined.

The overall conclusion of this research is that the replacement method and direction as well as the circuit congestion have no predictable impact on the reconfiguration time at any particular location on the FPGA. It is possible that there are other architectural factors which were not used in this study that affect the reconfiguration time. However, the factors used for this research do not statistically demonstrate one replacement method is consistently better than another at any particular location on the Spartan 3 chip.

5.4 Significance of Research

This is the first study of the effects of replacement method and circuit congestion on reconfiguration time of an FPGA. Previous research on fault tolerance using FPGAs focused mainly on algorithm development to achieve reconfiguration through the efficient use of spare CLBs and routing resources in the FPGA. There has been no attempt to determine the best reconfiguration in terms of reconfiguration time.

The primary significance of this research lies in the area of reconfiguration algorithm development. For a fault tolerant system using reconfiguration of an FPGA, an algorithm to control the circuit reconfiguration is free to use whatever replacement method is available without considering circuit congestion, column or row movement, or replacement direction. The most convenient or efficient method in terms of FPGA resources or algorithm function can be used without introducing significant overhead in terms of reconfiguration time.

5.5 Recommendations for Future Research

5.5.1 Other FPGAs and Programming Interfaces. An obvious variation of this study involves performing the same measurements as this research but on different FPGAs. This study used a Xilinx Spartan 3 FPGA. Using some other FPGA with a different architecture than the Spartan 3 may prove the hypotheses true due to some unknown factor not studied here.

This study used the JTAG interface to program the FPGA. While this is not the fastest or most efficient way to reconfigure, it was acceptable since this research was not focused on the magnitude of the reconfiguration time but on the comparison of the reconfiguration time for the four methods studied. Using the Xilinx Virtex 2 FPGA would allow use of an internal configuration access port (ICAP). The configuration bit file could be stored on the FPGA in block memory. Upon receiving a reconfiguration command, the bit file would be applied to the ICAP for reprogramming the FPGA [FHA03]. This would be a more efficient and faster method of accomplishing reconfiguration and reduces the overhead external to the FPGA.

5.5.2 Automation. A method for automating the control functions of the CRS would be a logical next step. In this study, generation of the configuration bit files and control of the reconfiguration process were done off-line. Ideally, upon notification of a fault and its location by some fault detection mechanism, the control circuit would dynamically generate the proper configuration bit file and initiate the reconfiguration of the FPGA. The ability to generate the configuration bit files dynamically would eliminate pre-fault generation and storage of all possible variations of bit files necessary to reconfigure the FPGA in response to a fault in any CLB.

5.5.3 Bit File Manipulation. Another area for future study is the manipulation of the original configuration bit file to cause the movement of a CLB when reconfiguration becomes necessary. Knowing which bits to change in the configuration bit file could facilitate process automation as discussed above.

Appendix A. Experimental Configuration

The experimental configuration required two FPGA development boards. One was programmed with the circuit being reconfigured while the other was programmed with the timer circuit used to measure the reconfiguration time.

The circuit being reconfigured consists of a 4-input LUT controlled by a state logic module as shown in Figure A.1. The circuit also contains a post configuration processing module. The LUT and its post configuration processing module use three of the four available slices in a CLB. During the reconfiguration process, the location of the LUT is moved left or right one column or up or down one row.

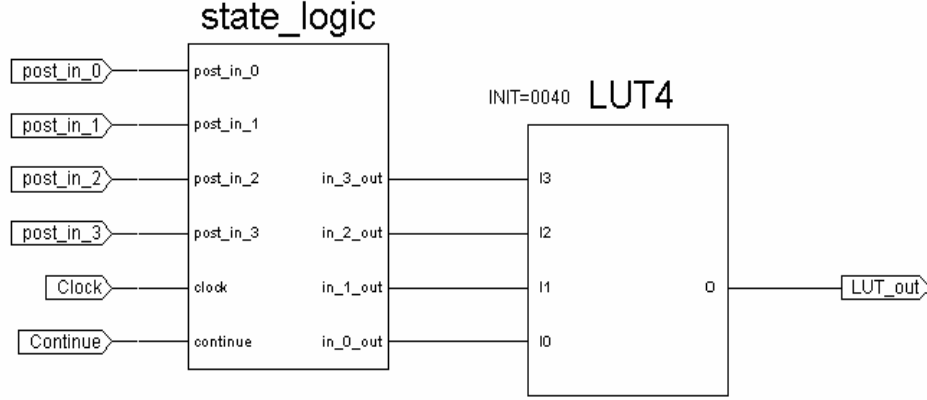


Figure A.1: Schematic of the 4-input LUT and the associated state logic in the CRS

The inputs to the LUT are from the state logic module. The state logic module determines whether to hold the LUT at its last state before reconfiguration or to accept external inputs for continued circuit operation. After reconfiguration, the LUT inputs are held in their previous states until the continue signal is asserted. At that point, the LUT inputs are determined through normal circuit functions.

The post configuration processor is shown in Figure A.2. It is used to simulate external circuit function by providing changing inputs to the LUT after reconfiguration. The outputs of the post configuration processor are provided to the state logic module where the LUT input is chosen.

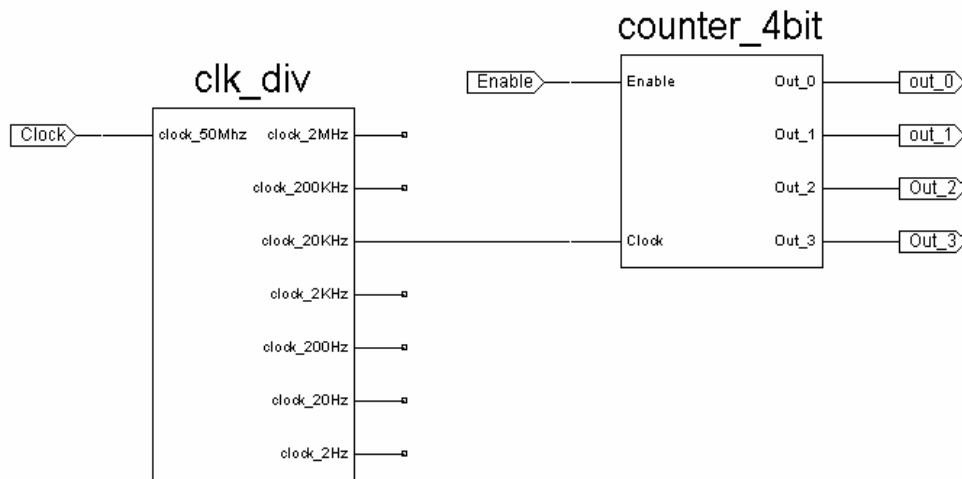


Figure A.2: The post configuration processor of the CRS simulates an external circuit by providing inputs to the LUT after reconfiguration

Configuration and reconfiguration is carried out through the JTAG port on the development board. A base circuit was laid out using the FPGA editor tool in Xilinx Project Navigator. The base configuration bit file was generated from this layout. The LUT was then moved according to the replacement method being tested and a partial bit file was generated to reconfigure the FPGA accordingly. Measurements were performed by first configuring the FPGA with the base configuration bit file and then reconfiguring it with the partial bit file corresponding to the proper movement of the LUT.

The VHDL code to implement the circuit which is being moved by the CRS is shown below.

```
-- Description: This module implements a state initialization for a
-- 4 input LUT after reconfiguration. The state is held until a
-- "continue" signal is received at which time the reconfigured
-- circuit resumes normal operation.
-- Author: Jason Ives
-- Date: November 21, 2005
```

```

entity state_logic is
    Port ( post_in_0      : in  std_logic;
           post_in_1      : in  std_logic;
           post_in_2      : in  std_logic;
           post_in_3      : in  std_logic;
           clock           : in  std_logic;
           continue        : in  std_logic;
           in_0_out        : out std_logic;
           in_1_out        : out std_logic;
           in_2_out        : out std_logic;
           in_3_out        : out std_logic);
end state_logic;

architecture Behavioral of state_logic is
    signal output_0 : std_logic;
    signal output_1 : std_logic;
    signal output_2 : std_logic;
    signal output_3 : std_logic;
begin
    process
    begin
        wait until clock'event and clock = '1';
        if continue = '1' then
            output_0 <= post_in_0;
            output_1 <= post_in_1;
            output_2 <= post_in_2;
            output_3 <= post_in_3;
        else
            output_0 <= '0';
            output_1 <= '1';
        end if;
    end process;
end Behavioral;

```



```

        output_2 <= '1';
        output_3 <= '0';
    end if;

end process;

in_0_out <= output_0;
in_1_out <= output_1;
in_2_out <= output_2;
in_3_out <= output_3;
end Behavioral;

```

Because reconfiguration is accomplished through the JTAG port, the JTAG test clock, TCK, is used to measure the reconfiguration time. The TCK signal controls the JTAG configuration process and is used to clock the configuration bits into the configuration registers of the FPGA. Thus, the size of the configuration bit stream determines how long the TCK signal is active during reconfiguration. The timer circuit simply measures the time the TCK signal is active.

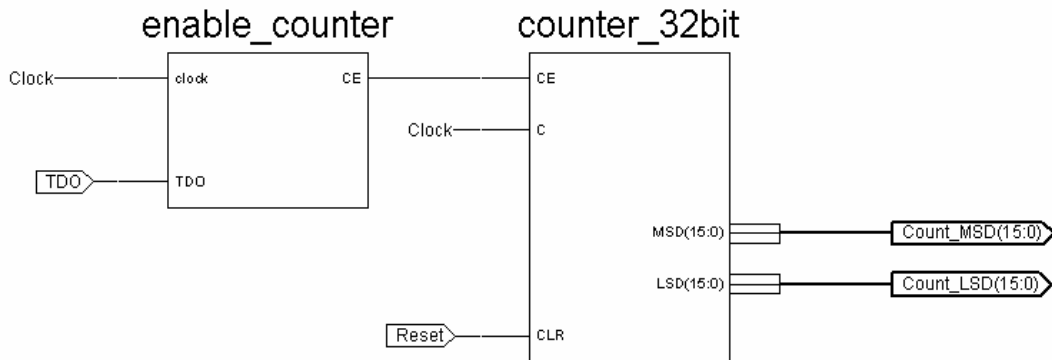


Figure A.3: Schematic of the timer circuit used to measure the reconfiguration time of the CRS

The timer circuit, shown in Figure A.3 counts the 50 MHz system clock to measure the reconfiguration time. The 8-bit hex count is converted to seconds for analysis. The 32-bit counter module in the timer circuit is a standard logic block available in most VHDL synthesis libraries. The VHDL code used to implement the

enable counter module is shown here because it is used to start and stop the 32-bit counter. Thus, its function defines the reconfiguration time in terms of the TCK signal.

```
-- Description:  This module generates a counter enable signal
-- for the 32 bit counter based on the TCK level.
-- CE is high if TCK is high or if there is less than
-- 50ms between high-low-high transitions.
-- Author:  Jason Ives
-- Date:  December 12, 2005
entity enable_counter is
    Port ( clock      : in std_logic;
          TCK         : in std_logic;
          CE          : out std_logic);
end enable_counter;
architecture Behavioral of enable_counter is
begin
    process (TCK, clock)
        variable count: integer range 0 to 6000000;
    begin
        if TCK = '1' then
            count := 0;
            CE <= '1';
        elsif clock'EVENT and clock = '1' and TCK = '0' then
            count := count + 1;
        end if;
        if count >= 5000000 then
            CE <= '0';
        end if;
    end process;
end;
```

`end Behavioral;`

As can be seen in the code segment, the CE signal stays high for 100 ms (5,000,000 clock cycles) after the end of the TCK signal in order to properly capture the entire time the TCK clock is active. This extra 100 ms was subtracted from the measured time before the data was analyzed.

Various manufacturer's data sheets, application notes and user's guides were referenced to arrive at the experimental setup [[DS005](#), [Xil05a](#), [Xil04b](#), [Xil04a](#), [Xil05b](#), [Xil03](#), [Xil](#)].

Appendix B. Data

This appendix contains the measured data from the experiments.

Table B.1: Measured reconfiguration time in seconds for location A1.

Location	Type	Column Left	Column Right	Row Up	Row Down
A1	Clear	1.24636805	0.76017080	1.39971985	1.24320360
A1	Clear	1.24581110	0.76094965	1.39897785	1.23552005
A1	Clear	1.24877890	0.76158465	1.39920090	1.23537725
A1	Clear	1.24964040	0.76002830	1.39831295	1.23613720
A1	Clear	1.24796065	0.76243050	1.39780570	1.23411275
A1	Clear	1.26277345	0.76025410	1.39802665	1.23628230
A1	Clear	1.24575640	0.78590480	1.39880665	1.23667730
A1	Clear	1.26117040	0.76150540	1.39984740	1.23503050
A1	Clear	1.24581950	0.76319045	1.42475295	1.23703840
A1	Clear	1.24648325	0.75927490	1.39563905	1.23813770

Table B.2: Measured reconfiguration time in seconds for location A2.

Location	Type	Column Left	Column Right	Row Up	Row Down
A2	Clear	1.57183035	1.23344700	1.24018245	1.26046215
A2	Clear	1.57402110	1.23849330	1.23855845	1.26060400
A2	Clear	1.57297760	1.23722320	1.23335145	1.26262310
A2	Clear	1.59423330	1.23534295	1.23735940	1.26223260
A2	Clear	1.57696900	1.23612305	1.23810460	1.26193770
A2	Clear	1.57608170	1.24109275	1.23709890	1.26235110
A2	Clear	1.57240005	1.23757815	1.24172860	1.25932975
A2	Clear	1.57702395	1.23560100	1.23789465	1.26173720
A2	Clear	1.57875675	1.23795535	1.23427620	1.26116870
A2	Clear	1.57166580	1.24232165	1.23606340	1.26061145

Table B.3: Measured reconfiguration time in seconds for location A3.

Location	Type	Column Left	Column Right	Row Up	Row Down
A3	Clear	1.55940295	1.73394740	0.92148460	1.59483525
A3	Clear	1.55885685	1.73361475	0.92304335	1.59677040
A3	Clear	1.55745905	1.73288750	0.92289510	1.59291450
A3	Clear	1.55937180	1.73358445	0.92299425	1.59421160
A3	Clear	1.56080745	1.73419210	0.92221470	1.59646045
A3	Clear	1.55910860	1.73670675	0.93180405	1.59256785
A3	Clear	1.55850830	1.73401065	0.91962645	1.60825400
A3	Clear	1.56444585	1.73346315	0.92055240	1.59568410
A3	Clear	1.56017480	1.73470950	0.92094490	1.59978695
A3	Clear	1.55938505	1.73708675	0.92130665	1.60267025

Table B.4: Measured reconfiguration time in seconds for location B1.

Location	Type	Column Left	Column Right	Row Up	Row Down
B1	Congested	1.08541075	1.24491515	1.42208240	1.09885605
B1	Congested	1.08530395	1.24937320	1.41999810	1.09663065
B1	Congested	1.08435690	1.25683520	1.42307575	1.09455895
B1	Congested	1.08487940	1.24544245	1.42597455	1.09614405
B1	Congested	1.08712280	1.24536880	1.42429500	1.09647675
B1	Congested	1.08456830	1.24678670	1.41851990	1.09660740
B1	Congested	1.08514385	1.24565420	1.42129145	1.09731210
B1	Congested	1.08561960	1.25495190	1.41862885	1.09467845
B1	Congested	1.08487830	1.25518535	1.42154175	1.09623050
B1	Congested	1.08516245	1.24528505	1.43049115	1.09437020

Table B.5: Measured reconfiguration time in seconds for location B2.

Location	Type	Column Left	Column Right	Row Up	Row Down
B2	Congested	1.24729425	1.40874055	1.08523645	1.08386435
B2	Congested	1.24897170	1.40993540	1.08751360	1.08601805
B2	Congested	1.24560145	1.40830985	1.08460850	1.08582840
B2	Congested	1.24348500	1.41089040	1.08319600	1.08684330
B2	Congested	1.24803445	1.40777555	1.08351315	1.08463975
B2	Congested	1.24610135	1.40929320	1.08474040	1.08477880
B2	Congested	1.24817685	1.40877120	1.08269175	1.08310460
B2	Congested	1.24759085	1.40823565	1.08738355	1.08521595
B2	Congested	1.24691425	1.41033870	1.08461880	1.08619340
B2	Congested	1.24432985	1.40732780	1.08622620	1.08308185

Table B.6: Measured reconfiguration time in seconds for location B3.

Location	Type	Column Left	Column Right	Row Up	Row Down
B3	Congested	1.26828126	1.27202474	0.94569462	0.78001304
B3	Congested	1.28955354	1.26315608	0.95081762	0.76676186
B3	Congested	1.27211944	1.26641872	0.95132944	0.76907878
B3	Congested	1.27598262	1.27912006	0.94634830	0.79129678
B3	Congested	1.27429994	1.31334406	0.96040870	0.77403574
B3	Congested	1.26880652	1.26016820	0.97178832	0.79209702
B3	Congested	1.26688860	1.26602624	0.94416720	0.76389944
B3	Congested	1.26676400	1.26156258	0.96818026	0.76852242
B3	Congested	1.26984670	1.26812550	0.94407888	0.76755902
B3	Congested	1.29897952	1.27332106	0.94476348	0.77474790

Bibliography

- AL81. T. Anderson and P. Lee. *Fault Tolerance Principles and Practice*. Prentice Hall, 1981.
- CC95. R. Cuddapah and M. Corba. Reconfigurable logic for fault tolerance. In W. Moore and W. Luk, editors, *Field-Programmable Logic and Applications, 1995. Proceedings. 5th International Workshop on*, number 975 in Lecture Notes in Computer Science, pages 380–388, 1995.
- CH02. Katherine Compton and Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.
- CT02. R.O. Canham and A.M. Tyrrell. A multilayered immune system for hardware fault tolerance within an embryonic array. In *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, pages 3–11, 2002.
- DP94. S. Durand and C. Piguet. FPGAs with self-repair capabilities. In *FPGAs, 1994. Proceedings. ACM International Workshop on*, 1994.
- DS005. Spartan-3 FPGA family: Complete data sheet, 2005.
- EB97. John M. Emmert and Dinesh Bhatia. Partial reconfiguration of FPGA mapped designs with applications to fault tolerance and yield enhancement. In *FPL '97: Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pages 141–150, London, UK, 1997. Springer-Verlag.
- Els03. Khaled Elshafey. Embedding fault tolerance via reconfiguration in configurable systems. In *ICM 2003, Proceedings of the*, pages 370–373, December 2003.
- ESSA00. J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici. Dynamic fault tolerance in FPGAs via partial reconfiguration. In *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, pages 165–174, 2000.
- FHA03. R.J. Fong, S.J. Harper, and P.M. Athanas. A versatile framework for FPGA field updates: an application of partial self-reconfiguration. In *Rapid Systems Prototyping, 2003. Proceedings. 14th IEEE International Workshop on*, pages 117–123, 2003.
- GASF03. M.G. Gericota, G.R. Alves, M.L. Silva, and J.M. Ferreira. Run-time management of logic resources on reconfigurable systems. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 974–979, 2003.

- Hat93. Sakurai-T. Nogami K. Sawada K. Takahashi M. Ichida M. Uchida M. Yoshii I. Kawahara Y. Hibi T. Saeki Y. Muroga H. Tanaka A. Kanzaki K. Hatori, F. Introducing redundancy in field programmable gate arrays. In *Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993*, pages 7.1.1–7.1.4, 1993.
- Hau98. S. Hauck. The roles of FPGAs in reprogrammable systems. *Proceedings of the IEEE*, 86(4):615–638, 1998.
- HD98. F. Hanchek and S. Dutt. Methodologies for tolerating cell and interconnect faults in FPGAs. *Computers, IEEE Transactions on*, 47(1):15–33, 1998.
- HKSW98. James R. Heath, Philip J. Kuekes, Gregory S. Snider, and R. Stanley Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280:1716–1721, 1998.
- HTA94. N.J. Howard, A.M. Tyrrell, and N.M. Allinson. The yield enhancement of field-programmable gate arrays. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(1):115–123, 1994.
- KDFJ89. V. Kumar, A. Dahbura, F. Fischer, and P. Juola. An approach for the yield enhancement of programmable gate arrays. In *Computer-Aided Design, 1989. ICCAD-89. Digest of Technical Papers., 1989 IEEE International Conference on*, pages 226–229, 1989.
- Kha02. Jamil Khatib. Introduction to programmable logic devices. http://www.geocities.com/jamilkhatib75/fpga/fpga_intro.html, 2002.
- KI94. J.L. Kelly and P.A. Ivey. Defect tolerant SRAM based FPGAs. In *FPGAs, 1994. Proceedings. ACM International Workshop on*, pages 479–482, 1994.
- Kwi97. K. Kwiat. Dynamically reconfigurable FPGA based multiprocessing and fault tolerance. Technical Report RL-TR-96-279, Rome Labs, 1997.
- LCR03. F. Lima, L. Carro, and R. Reis. Designing fault tolerant systems into SRAM-based FPGAs. In *Design Automation Conference, 2003. Proceedings*, pages 650–655, 2003.
- LMSP98a. J. Lach, W.H. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant FPGA systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 6(2):212–221, 1998.
- LMSP98b. John Lach, William H. Mangione-Smith, and Miodrag Potkonjak. Efficiently supporting fault-tolerance in FPGAs. In *FPGA '98: Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, pages 105–115, New York, NY, USA, 1998. ACM Press.

- LMSP99. J. Lach, W.H. Mangione-Smith, and M. Potkonjak. Algorithms for efficient runtime fault recovery on diverse FPGA architectures. In *Defect and Fault Tolerance in VLSI Systems, 1999. DFT '99. International Symposium on*, pages 386–394, 1999.
- LMSP00. J. Lach, W.H. Mangione-Smith, and M. Potkonjak. Enhanced FPGA reliability through efficient run-time fault reconfiguration. *Reliability, IEEE Transactions on*, 49(3):296–304, 2000.
- Max04. C. Maxfield. *The Design Warriors Guide to FPGAs*. Elsevier, 2004.
- May97. C. Mayer. A reconfigurable superscalar architecture. Master’s thesis, Air Force Institute of Technology, 1997.
- McF94. C. McFarland. Computer subsystem. <http://www.tsgc.utexas.edu/archive/subsystems/>, 1994.
- MD99. N.R. Mahapatra and S. Dutt. Efficient network-flow based techniques for dynamic fault reconfiguration in FPGAs. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 122–129, 1999.
- MGM⁺96. D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, S. Durand, P. Marchal, and P. Nussbaum. Embryonics: A new family of coarse-grained field-programmable gate array with self-repair and self-reproducing properties. In *Circuits and Systems, 1996. ISCAS '96., 'Connecting the World', 1996 IEEE International Symposium on*, volume 4, pages 25–28 vol.4, 1996.
- MHS⁺04. Subhasish Mitra, W.-J. Huang, N.R. Saxena, S.-Y. Yu, and E.J. McCluskey. Reconfigurable architecture for autonomous self-repair. *Design & Test of Computers, IEEE*, 21(3):228–240, 2004.
- ML96. A. Mathur and C.L. Liu. Timing driven placement reconfiguration for fault tolerance and yield enhancement in FPGAs. In *European Design and Test Conference, 1996. ED&TC 96. Proceedings*, pages 165–169, 1996.
- Nei03. Mohamad R. Neilforoshan. Fault tolerant computing in computer design. *Journal of Computing in Small Colleges*, 18(4):213–220, 2003.
- Nel90. Victor P. Nelson. Fault-tolerant computing: Fundamental concepts. *IEEE Computer*, pages 20–25, 1990.
- NSF01. K. Nikolic, A. Sadek, and M. Forshaw. Architectures for reliable computing with unreliable nanodevices. In *Nanotechnology, 2001. Proceedings of the 2001 1st IEEE Conference on*, pages 254–259, 2001.
- OT97. C. Ortega and A. Tyrrell. Biologically inspired reconfigurable hardware for dependable applications. In *Hardware Systems for Dependable Applications, 1997. IEE Colloquium on*, pages 3/1–3/4, 1997.

- Pra05. D.K. Pradhan. Fundamental concepts of redundancy for fault tolerance. http://www.cs.bris.ac.uk/Teaching/Resources/COMS30125/lecture_3.ppt, September 2005.
- REGSV93. J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli. Architecture of field-programmable gate arrays. *Proceedings of the IEEE*, 81(7):1013–1029, 1993.
- SP03. A.P. Shanthi and R. Parthasarathi. Exploring FPGA structures for evolving fault tolerant hardware. In *Evolvable Hardware, 2003. Proceedings. NASA/DoD Conference on*, pages 174–181, Jul 2003.
- Tor02. Jim Torresen. Reconfigurable logic applied for designing adaptive hardware systems. In *Book of abstracts of the International Conference on Advances in Infrastructure for Electronic Business, Education, Science, and Medicine on the Internet*, 2002.
- XAR03. Xilinx answer record 9803, March 2003.
- Xil. Xilinx. *Libraries Guide*.
- Xil03. Xilinx. Configuration quick start guidelines, 2003.
- Xil04a. Xilinx. Spartan-3 advanced configuration architecture, 2004.
- Xil04b. Xilinx. Two flows for partial reconfiguration: Module based or difference based, 2004.
- Xil05a. Xilinx. Spartan-3 Starter Kit Board User Guide, 2005.
- Xil05b. Xilinx. Using the ISE design tools for Spartan-3 generation FPGAs, 2005.
- XSHL99. Jian Xu, Paifa Si, Weikang Huang, and F. Lombardi. A novel fault tolerant approach for SRAM-based FPGAs. In *Dependable Computing, 1999. Proceedings. 1999 Pacific Rim International Symposium on*, pages 40–44, 1999.

Vita

Capt Jason L. Ives was born October 3, 1968 in Tucson, Arizona. He graduated from Flowing Wells High School, Tucson, Arizona in May 1986. In April 1989, Capt Ives enlisted in the United States Air Force and was trained as an air traffic control radar maintenance specialist. Upon graduation from technical training, he was assigned to Tempelhof Central Airport, Berlin, Germany and, later, to Beale Air Force Base, California. In December 1997, Capt Ives returned to Tucson to attend the University of Arizona under the Airman Education and Commissioning Program. After graduating in December 2000, he earned a commission through the Air Force Officer Training School. He was subsequently assigned to the National Security Agency, Fort George G. Meade, Maryland. In July 2004, Capt Ives was assigned to Wright Patterson Air Force Base as a student at the Air Force Institute of Technology. After graduation from AFIT, Capt Ives will report to the sensors directorate of the Air Force Research Laboratory at Wright Patterson.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 23-03-2006		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) Sept 2004 — Mar 2006		
4. TITLE AND SUBTITLE Evaluation of a Field Programmable Gate Array Circuit Reconfiguration System				5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Jason L. Ives, Capt, USAF				5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GE/ENG/06-26		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/Space Electronics and Protection Branch Mr. Ken Hunt 3550 Aberdeen Ave SE, Bldg 891 Kirtland AFB, NM 87117-5776 (505) 846-4959				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/VSSSE 11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT Approval for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT This research implements a circuit reconfiguration system (CRS) to reconfigure a field programmable gate array (FPGA) in response to a faulty configurable logic block (CLB). It is assumed the location of the fault is known and the CLB is moved according to one of four replacement methods: column left, column right, row up, and row down. Partial reconfiguration of the FPGA is done through the JTAG port to produce the desired logic block movement. The time required to accomplish the reconfiguration is measured for each method in both clear and congested areas of the FPGA. The measured data indicates there is no consistently better replacement method regardless of the circuit congestion or location within the FPGA. Thus, given a specific location in the FPGA, there is no preferred replacement method that will result in the lowest reconfiguration time.						
15. SUBJECT TERMS field programmable gate array, fault tolerance, reconfiguration						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES	
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	 UU		 76	
					19a. NAME OF RESPONSIBLE PERSON Dr. Rusty O. Baldwin (ENG)	
					19b. TELEPHONE NUMBER (include area code) (937) 255-6565, ext 4445	